AFRL-IF-RS-TR-2007-113
Final Technical Report
April 2007

A MODEL OF TRUST FOR DEVELOPING TRUSTWORTHY SYSTEMS FROM UNTRUSTWORTHY ACTORS

Colorado State University

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

AFRL-IF-RS-TR-2007-113 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/ /s/

ROBERT VAETH
Work Unit Manager

WARREN H. DEBANY, Jr.
Technical Advisor, Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
APR 2007	Final	May 03 – Nov 06

4. TITLE AND SUBTITLE

A MODEL OF TRUST FOR DEVELOPING TRUSTWORTHY SYSTEMS FROM UNTRUSTWORTHY ACTORS

5a. CONTRACT NUMBER

5b. GRANT NUMBER
F30602-03-1-0101

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Indrajit Ray and Indrakshi Ray

5d. PROJECT NUMBER
FAA6

5e. TASK NUMBER
2N

5f. WORK UNIT NUMBER
PR

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Colorado State University
601 S Howes St.
Fort Collins, CO 80523-2002

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFGB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2007-113

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 07-199

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The objective of this effort is to develop a new model of trust that allowed one to reason about trust relationships in information systems with special emphasis on trust as it related to integrity and availability. The project has produced the following results. 1. It has proposed a formal model to assess multiple levels of trust that is more inclusive than the current binary models. A major strength of the model is that it is more in keeping with the social models of trust used by policy makers. 2. It has defined a notion of degrees of trust and proposed expressions and procedures to evaluate and establish the degree of trust of different systems. 3. It has defined procedures to compare information at different degrees of trust. 4. It has developed procedures to determine the trust level of composed information. 5. It has formulated processes and procedures to manage trust relationships.

15. SUBJECT TERMS
Trust model, trust relationships, information systems, integrity, availability, formal model

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert Vaeth
U	U	U	UL	91	19b. TELEPHONE NUMBER (*Include area code*)

Contents

List of Figures

Executive Summary

The notion of *trust* is widely used in secure information systems. For example, “trusted computing base” refers to the hardware and software that make up the security of a system; a “trusted system” is one that is believed to be secure against relevant attacks, and so on. However, until recently, there were no accepted formalism or techniques for the specification of trust and for reasoning about it. Secure systems had been built under the premise that concepts like “trusted” or “trustworthiness” were well understood, unfortunately without even agreeing on what “trust” means, how to measure it, how to compare two trust values and how to compose the same. There was a lack of a comprehensive mathematical framework for quantifying the amount of trust that can be placed on complex systems that had been built from smaller components. This led to considerable degrees of inferential ambiguities when security related decisions had to be made based on trust. Additionally, most researchers had addressed trust related issues from the perspective of access control in the confidentiality context.

The objective of this effort was to develop a new model of trust that allowed one to reason about trust relationships in information systems with special emphasis on trust as it related to integrity and availability. The project has produced the following results.

1. It has proposed a formal model to assess multiple levels of trust that is more inclusive than the current binary models. A major strength of the model is that it is more in keeping with the social models of trust used by policy makers.

2. It has defined a notion of degrees of trust and proposed expressions and procedures to evaluate and establish the degree of trust of different systems.

3. It has defined procedures to compare information at different degrees of trust.

4. It has developed procedures to determine the trust level of composed information.

5. It has formulated processes and procedures to manage trust relationships.

Various aspects of the model were subjected to peer-review. A number of technical papers in reputable international conferences have resulted in the process. A technical paper that details the complete model is currently being reviewed by the editorial board of a top-level journal.

The project investigated the challenges of access control in open and distributed environments in an attempt to determine how this area can benefit from the new trust model. An example of such an environment is the NAS system of the FAA. Preliminary results from the investigation shows promise. Results have been published in major conferences. Further investigation is being done.

The project team has identified a number of open issues that, if addressed, will enhance the expressive power of the model and make it more usable. Among these are (i) the proper formulation of trust context, (ii) the ability to extrapolate trust relationships and (iii) the definition of and reasoning with trust chains. We are looking forward to continued support from the AFRL and the FAA for this purpose.

Chapter 1

Introduction

Information technology is increasingly driven by the requirements of confidentiality, integrity, availability, usability and digital rights management of systems and information resources. To ensure that information systems behave according to stated requirements, proper techniques and procedures need to be used in designing and implementing the system. A lot of research has been done in developing such techniques and in evaluating the degree to which the system will behave in isolation according to these stated requirements. However, a systems behavior is frequently dependent on other systems. To measure the confidence that users have in their decision to accept the assumed or measured amount of proper behavior of the system in the face of influence from other systems, the notion of *trust* is widely used. However, there are no accepted formalisms or techniques for the specification and measurement of trust and for reasoning about trust. For the most part, trust is considered to be a binary entity; confidence is measured in terms of either total trust or no trust.

This binary model of trust differs considerably in semantics from the social models of trust used by policy makers. In sociology, trust means the assured reliance on the character, ability, strength or truth of someone or something. This assurance level can be of different degrees, leading to entities being labeled trusted to various levels. Within the domain of computer security, however, the term *trusted* is used strictly to indicate the successful evaluation of authorization requirements for security-critical actions. This creates inferential ambiguities, for example in the composition of information gathered from different sources or in the composition of systems that involve interaction between human and computational devices. As systems evolve, a significant chasm emerges between our sociological perception of trust or trustworthiness and information technology's view of trust.

Consider a collaborative network defense system deployed within a local area network (LAN)

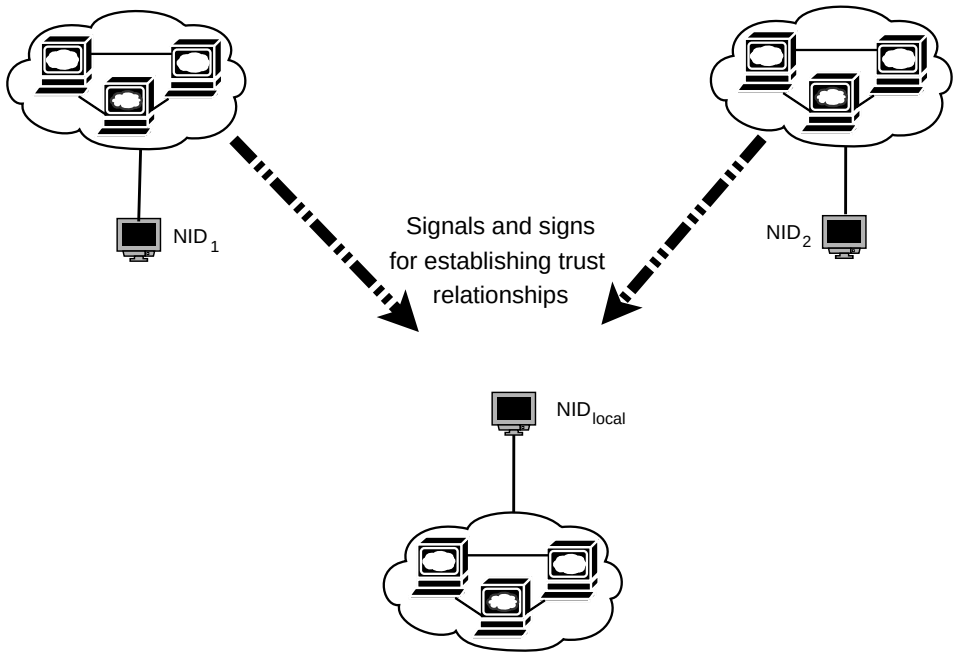


Figure 1.1: Mutual assessment of trustworthiness in a collaborative system

as shown in figure 1.1. A major component of the system is a distributed network intrusion detection (NID) module that can gather information from other network intrusion detection systems deployed elsewhere on LANs that are under separate administrative controls. The NID module monitors the local network for possible intrusion scenarios and also seeks information about intrusion alerts from some of the other similar modules deployed in other LANs. The module then analyzes the information and advises the local LAN administrator about the possibility of a network attack in the near future. The NIDs modules belonging to other LANs behave in a similar manner. Now the local NID (call it NID_{local}) can trust the information that it gathers from the local LAN. However, it may be too naïve for the local NID to trust the others completely. Here are several situations where this will be the case. Assume that one of the remote NIDs bears a certificate from an independent testing agency that attests to the fact that the NID application was submitted by its developer for testing and has been found to be free of malicious code and other defects. However, the certification agency may not have followed proper procedure in the certification process; the certification agency's own credentials may have been revoked but the information may have not trickled down to the end user; or the developer may have tweaked with the software after the certification. Under such circumstances although the certificate attests to the competence of the system, it is not authoritative enough. We are left with no rational approach for answering the following questions: (1) What expectations can the LAN administrator have about the usefulness of the composed information? (2) What critical activities can the administrator use the information

for without much problem? (3) What are the critical activities that the administrator does not want to fulfill using this information?

These problems arise because there are two aspects to the evaluation of trustworthiness of systems. The first aspect is determining whether a system is competent or not. The second aspect is determining the firmness in the belief about our evaluation of the system's competency. A number of works have looked into the former aspect; however the latter is neglected, at least within the computer security and dependability area. We address this second aspect in the current work.

The above observations motivate us to propose a new "Vector" model of trust. In this trust is a measurable entity which can have different degrees. We specify trust as a vector of numeric values. Each element of the vector influences the value of trust. We identify four such parameters in our model. We propose methods to determine the values corresponding to these parameters. Substituting values for each of these parameters in the trust vector provides a value for trust. This vector now represents trust of a certain degree. To render the concept of different degrees of trust more intuitive, we associate a numeric value in the range $[-1,1]$ with the trust vector. The value in the positive region of this range is used to express trust and that in the negative region is used to express distrust. Neutrality about trust and distrust is expressed using the value zero. We also use a special value, denoted by '$\perp$' to represent "lack of information" about trust. We define operators to map a trust vector to a trust value and vice versa. Next, we investigate the dynamic nature of trust – how trust (or distrust) changes over time. We observe that trust depends on trust itself – that is a trust relationship established at some point of time in the past influences the computation of trust at the current time. We formalize this notion in our model. Defining comparison operator for trust vectors, allows us to make a decision about relative "trustworthiness" of two or more entities. Finally, we define a mechanism to combine trusts of different degrees to form a single trust relationship. This helps us model the evolution of a trust relationship between a group of trusters and a group of trustees.

Using our model, on the other hand, we can analyze the situation presented in figure 1.1 as follows. Since the crux of the problem lies in an ability to determine trust among the various agents, we propose in our model an approach to determine how much trust the local NID (NID_{local}) can have on the information collected from other NIDs (say, NID_1 and NID_2).

The system initiates with NID_{local} maintaining a neutral position about the trustworthiness of NID_1 and NID_2. As time progress, NID_{local} will gradually begin to establish trust relationships with the other two. The degree of trust that NID_{local} establishes with another NID will depend on different factors. For example, NID_{local} may become aware that the other NIDs are exactly the same applications as itself. This *knowledge* together with a trust policy that establishes guidelines

on how to use the knowledge helps NID_{local} to gain some trust on the other two NIDs. Further, over a period of time NID_{local} begins to have positive *experiences* with NID_2 and some positive and some negative experiences with NID_1. (Note that the perception of positive experience or negative experience is quite subjective. We leave it like that because, after all, the notion of trust is also very subjective.) At some point then the NID_{local} will (perhaps) evaluate with the help of our model that NID_1 is trusted to a degree of 0.25 and trust NID_2 to degree of 0.75. The local NID will then consult its policy base to determine how an information corresponding to this trust value should be used.

The above approach provides a more rational way to evaluating the trustworthiness of systems. It has got a number of advantages. The biggest is that the model allows a truster to determine a trust level even in the face of incomplete information about factors that the truster uses to judge the trustworthiness of the trustee. In the worst case, for instance, the model will compel the truster to take a *neutral* position. The second major advantage is that the model takes into consideration individual perceptions about trust by way of the notion of a trust evaluation policy. A third major advantage is that this trust model can neutralize to a great extent the scenario where a recommender tells a lie to gain an unfair advantage without using complex game theoretic approaches.

The rest of the report is organized as follows. We present the Vector model of trust in chapter 2. Chapter 3 describes the VTrust trust management system that has been developed to store, manage and manipulate trust relationships according to the Vector model. The VTrust trust management systems uses a language called TrustQL for accessing and manipulating the stored information. The complete syntax of the the TrustQL language is provided in chapter 4. In chapter 5 we discuss how to use the model to address a very important security problem, namely access control in open and distributed environments. Finally chapter 6 concludes the report with a discussion of some refinement and extensions of the model that we plan to address in the future.

Chapter 2

The Vector Model of Trust

2.1 Overview

Trust is modeled as a measurable entity that can have different degrees. We specify trust as a vector of numeric values. Each element of the vector influences the value of trust. We identify three such parameters in our model. We propose methods to determine the values corresponding to these parameters. Substituting values for each of these parameters in the trust vector provides a value for trust. This vector now represents trust of a certain degree. To render the concept of different degrees of trust more intuitive, we associate a numeric value in the range $[-1, 1]$ with the trust vector. The value in the positive region of this range is used to express trust and that in the negative region is used to express distrust. Neutrality about trust and distrust is expressed using the value zero. We also use a special value, denoted by '$\perp$' to represent "lack of information" about trust. We define operators to map a trust vector to a trust value and vice versa. Next, we investigate the dynamic nature of trust – how trust (or distrust) changes over time. We observe that trust depends on trust itself – that is a trust relationship established at some point of time in the past influences the computation of trust at the current time. We formalize this notion in our model. Defining comparison operator for trust vectors, allows us to make a decision about relative "trustworthiness" of two or more entities. Finally, we define a mechanism to combine trusts of different degrees to form a single trust relationship. This helps us model the evolution of a trust relationship between a group of trusters and a group of trustees.

The proposed model helps a user evaluate the amount of confidence she/he has in her/his decision to accept the assumed or measured amount of competency of a particular system when the system's behavior is influenced by other systems (including human beings). This allows the user to evaluate the risks involved in using the system in a better manner and thus design more trustworthy

systems. This is illustrated in the following discussion.

Consider the example of the network intrusion detection systems. To answer the questions posed earlier, we need to determine how much trust the local NID (NID_{local}) can have on the information collected from other NIDs (say, NID_1 and NID_2). In our model, the system initiates with NID_{local} maintaining a neutral position about trustworthiness of NID_1 and NID_2. As time progress, NID_{local} will gradually begin to establish trust relationships with the other two. The degree of trust that NID_{local} establishes with another NID will depend on different factors. For example, NID_{local} may become aware that the other NIDs are exactly the same applications as itself. NID_{local} may begin to have positive experiences with NID_2 and some positive and some negative experiences with NID_1. At some point then the NID_{local} will (perhaps) evaluate based on our model that NID_1 is trusted to a degree of 0.25 and trust NID_2 to degree of 0.75. Our model can further indicate that if information from NID_1 and NID_2 are combined, the resulting information can be trusted to a degree of 0.54 (say). Thus, if together NID_1 and NID_2 report that an attack is imminent, the current NID assumes that this information can be trusted to a degree of 0.54. It can then take a more effective decision (for example for allocating resources for defending against the attack).

2.2 Model Description

In our model we adopt the definition of trust as provided by Grandison and Sloman [5].

Definition 1 Trust is defined to be the firm belief in the competence of an entity to act dependably and securely within a specific context.

Definition 2 Distrust is defined as the firm belief in the incompetence of an entity to act dependably and securely within a specified context.

Although we define trust and distrust separately in our model, we allow the possibility of a neutral position where there is neither trust nor distrust. As we elaborate on the model this will become more clear.

Trust in our model is specified as a trust relationship between a truster – an entity that trusts the target entity – and a trustee – the target entity that is trusted. The truster is always an active entity (for example, a human being or a subject). The trustee can either be an active entity or a passive entity (for example, a piece of information or a software). We call an active entity an *actor* and a passive entity, a *component*. We use the following notation to specify a trust relationship – $(A \xrightarrow{c} B)_t^N$. It specifies A's *normalized* trust on B at a given time t for a particular context c. This

relationship is obtained from the simple trust relationship – $(A \xrightarrow{c} B)_t$ – by combining the latter with a normalizing factor. We also introduce a concept called the *value* of a trust relationship. This is denoted by the expression $\mathbf{v}(A \xrightarrow{c} B)_t^N$ and is a number in $[-1,1] \cup \{\perp\}$ that is associated with the normalized trust relationship. A trustee is completely trusted (or distrusted) if the value of the trust relationship is 1 (-1). If the value is in the range (0,1) the trustee is *semi-trustworthy*; if the value is in the range (-1,0) the trustee is *un-trustworthy*. The 0 value represents trust neutrality that is, the trustee is neither trustworthy nor un-trustworthy. The special symbol $\perp$ is used to denote the value when there is not enough information to decide about trust, distrust, or neutrality.

Definition 3 The *atomic purpose* of a trust relationship $(A \xrightarrow{c} B)_t$ is one of

1. **Access resources** The truster trusts a trustee to access and/or use in a proper manner some resources that the truster controls. Examples of this are reading/writing sensitive information, using properly copyrighted information and so on.
2. **Provide services** The truster trusts the trustee to provide a service that does not involve access to the truster's resources. Some examples of this will be hosting web services, provide a certification service and so on.
3. **Make decisions** The truster trusts the trustee in a decision making process. What kind decision making processes the truster wants to trust the trustee with depends on the truster's policy. An example of this will be deciding if a certificate is valid or not.

The truster may also trust the trustee for some combination of these atomic purposes. For example the truster may trust the trustee to provide a service and make decisions.

Definition 4 The *purpose* of a trust relationship is defined as follows.

1. An atomic purpose is a purpose of a trust relationship.
2. The negation of an atomic purpose, denoted by "not" atomic purpose, is a purpose.
3. Two purposes connected by the operator "and" form a purpose.
4. Two purposes connected by the operator "or" form a purpose.
5. Nothing else is a purpose.

Further, in our model of trust we are interested in ten *aspects* – availability, usability, reliability, safety, confidentiality, integrity, maintainability, accountability, authenticity and non-repudiability – of the trustee. Combining the concepts of trust purposes and trustee aspect, we define *trust context* as the interrelated conditions in which trust exists or occurs. For example, let a truster, *A*, trust a trustee, *B*'s reliability to provide a service and integrity to make a decision. The "reliability to provide a service and integrity to make a decision" is considered to be the trust context.

Definition 5 Let $\mathcal{S}$ denote the set of trust purposes and $\mathcal{A}$, the set of trustee aspects identified above. Then the *context, c(T),* of a trust relationship T is defined as follows:

1. A tuple of the form $< s_i, a_i >$ is a context where $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}$.
2. Two contexts connected by the operator "and" form a context.
3. Two contexts connected by the operator "or" is a context.
4. Nothing else is a context.

Definition 6 The *context function* $c(T)$ of a trust relationship T is a function that takes the trust relationship as the input and returns the context of that trust relationship.

2.2.1 Trust evaluation

Our trust model aims to provide the notion of a trust value to represent levels of trust. We face two choices for trust values – qualitative or quantitative. If qualitative values are used, then degree or level of trust can be expressed in terms of a set of discrete values such as high, medium or low. The advantage of such a scheme is that it is quite intuitive. However, the challenges are significant. First, it is rather difficult to define precisely the semantics of such levels; semantics across different systems can vary. Second, determining the appropriate number of such degrees for a particular system is not straightforward; in fact it can tend to be rather ad hoc. Third, determining how the degrees from different domains can be compared and combined is most difficult. Last but not the least, a problem with such discrete degrees is that it is not easy to represent ignorance or neutrality with respect to trust or distrust.

Quantifying trust through numeric values alleviate such problems. Moreover, mathematical operations on degrees of trust can be defined that allow proper comparison of degrees from different domains and combine them. This leads us to adopt numeric values for trust levels. Instead of limiting ourselves to a single value for trust (or distrust), we define a trust value in terms of a vector of numeric values. We use a vector so that we can specify the effects of the many different factors that influence trust. However, we also believe that there are times when a single numeric value is more intuitive than a vector of values – particularly when making comparisons in an informal manner. This leads us to define the notion of a "value" for a trust vector. It is a either single numeric value in the range $[-1, 1]$ or a special value $\perp$.

At this stage we point to two characteristics of trust (or distrust) that shapes our model. The first is the dynamic nature of trust. Trust changes over time. Even if there is no change in the underlying factors that influence trust over a time period, the value of trust at the end of the period is not the same as that at the beginning of the period. Irrespective of our initial trust or distrust decision,

over a period of time we gradually become non-decisive or uncertain about the trust decision. This leads us to claim that trust (and alternately distrust) decays over time - both tends towards a non-decisive value over time. The second characteristic is, what is often called the *propensity* to trust [5]. Given the same set of values for the factors that influence trust, two trusters may come up with two different trust values for the same trustee. We believe that there are two main reasons for this. First, during evaluation of a trust value, a truster may assign different weights to different factors that influence trust. The weights will depend on the trust evaluation policy of the truster. If two different trusters assign two different sets of weights, then the resulting trust value will be different. The second reason is applicable only when the truster is a human being and is completely subjective in nature – one person may be more trusting than another. We believe that this latter concept is extremely difficult to model in an objective manner. We choose to disregard this feature in our model and assume that all trusters are trusting in nature to the same extent. We capture the first factor using the concept of a *trust evaluation policy vector*, which is simply a vector of weight values. Using this weight vector on the simple trust relationship provides the normalized trust relationship.

We begin by identifying three different parameters that influence trust values – *experience*, *knowledge*, and *recommendation*.

Definition 7 The *experience* of a truster about a trustee is defined as the measure of the cumulative effect of a number of events that were encountered by the truster with respect to the trustee in a particular context and over a specified period of time.

The trust value of a truster on a trustee for some context can change because of the the truster's *experiences* with the trustee in the particular context. Consider the following scenario with our running example. NID_{local} has been witnessing that the information feed from NID_1 has been relevant for the past five months. Initially NID_{local} was neutral towards NID_1's information; however having benefited from it, NID_{local} now trusts NID_1 more to provide sound intrusion alerts in a timely manner.

A truster can categorize each experience about a trustee as *trust-positive*, *trust-negative* or *trust-neutral* experience. A trust-positive experience increases trust degree whereas a trust-negative experience diminishes trust degree. A trust-neutral event contributes neither way.

Definition 8 The *knowledge* of the truster regarding a trustee for a particular context is defined as a measure of the condition of awareness of the truster through acquaintance with, familiarity of or understanding of a science, art or technique.

The trust value of a truster on a trustee can change because of some *knowledge* that the truster comes to posses regarding the trustee for the particular context. Knowledge can be of two types – *direct knowledge* or *properties* and *indirect knowledge* or *reputation*.

Direct knowledge is one which the truster acquires by itself. It may be obtained by the truster in some earlier time for some purpose or, it may be a piece of information about the trustee for which the truster has a concrete proof to be true. Referring to our running example let NID_{local} want to establish a trust relationship with NID_1. NID_{local} begins by trying to identify what type of software NID_1 is. It determines that NID_1 is the same software as itself. This piece of information may enhance the trust of NID_{local}.

Indirect knowledge, on the other hand, is something that the truster does not acquire by itself. The source of indirect knowledge is the *reputation* of the trustee in the context. The truster may get the idea about the reputation of trustee from various sources like reviews, journals, news bulletin, people's opinion etc. With this reputation, without having specific information about the trustee, the truster can build an opinion about the trustee in the context. This piece of information is indirect because the truster has no way to determine the real truth behind the information. Finally, as with experience, we have *trust-positive*, *trust-negative*, and *trust-neutral* knowledge.

Definition 9 A *recommendation* about a trustee is defined as a measure of the subjective or objective judgment of a recommender about the trustee to the truster.

The trust value of a truster on a trustee can change because of a *recommendation* for the trustee. For example, a truster can ask someone close to him, who happens to know the trustee, about the latter's credibility (within the scope of the trust context). If that third person says "good words" about the trustee, the truster tends to have faith on the trustee. It is important to note that the importance of the judgment of the third person to the truster depends on how much the truster trusts the third person's ability to judge others. In our model we use the degree of trust between a truster and a recommender to evaluate the recommendation for the trustee. As before we can have a *trust-positive*, *trust-negative*, and a *trust-neutral* recommendation. Finally, recommendations can be obtained by the truster from more than one source and these together will contribute to the final trust relationship.

To compute a trust relationship we assume that each of these three factors is expressed in terms of a numeric value in the range $[-1,1]$ and a special value $\perp$. A negative value for the component is used to indicate the *trust-negative* type for the component, whereas a positive value for the component is used to indicate the *trust-positive* type of the component. A 0 (zero) value for the component indicates trust neutral. To indicate a lack of value due to insufficient information for

any component we use the special symbol $\perp$. If $\mathbb{R}$ is the set of real numbers, then (i) $a\cdot \perp = \perp \cdot a = \perp$, $\forall\, a \in \mathbb{R}$ (ii) $a + \perp = \perp + a = a$, $\forall\, a \in \mathbb{R}$ (iii) $\perp + \perp = \perp$ and $\perp \cdot \perp = \perp$

2.2.2 Evaluating experience

We model experience in terms of the number of events encountered by a truster, A, regarding a trustee, B in the context c within a specified period of time $[t_0, t_n]$. We assume that A has a record of the events since time t_0. An event can be trust-positive, trust-negative or, trust-neutral depending whether it contributes towards a trust-positive experience, a trust-negative experience or, a trust-neutral experience.

Let $\mathbb{N}$ denote the set of natural numbers. The set of time instances $\{t_0, t_1, \ldots, t_n\}$ is a totally ordered set, ordered by the temporal relation $\prec$, called the *precedes-in-time* relation, as follows: $\forall i, j \in \mathbb{N},\ t_i \prec t_j \Leftrightarrow i < j$. We use the symbol $t_i \preceq t_j$ to signify either $t_i \prec t_j$ or $t_i = t_j$. Let e_k denote the k^{th} event. Events happen at time instances. We define the concept *event-occurrence-time* as follows:

Definition 10 The *event-occurrence-time*, ET, is a function that takes an event e_k as input and returns the time instance, t_i at which the event occurred. Formally, $ET : e_k \rightarrow t_i$.

We divide the time period $[t_0, t_n]$ into a set $\mathcal{T}$ of n intervals, $[t_0, t_1], [t_1, t_2], \ldots, [t_{n-1}, t_n]$ such that for any interval $[t_i, t_j], t_i \prec t_j$. A particular interval, $[t_{k-1}, t_k]$, is referred to as the k^{th} interval. We extend the $\prec$ relation on $\mathcal{T}$ and the time intervals are also totally ordered by the $\prec$ relation as follows: $\forall i, j, k, l \in \mathbb{N},\ [t_i, t_j] \prec [t_k, t_l] \Leftrightarrow t_j \prec t_k$. The intervals are non-overlapping except at the boundary points, that is $\forall i, j, k, l \in \mathbb{N},\ [t_i, t_j] \cap [t_k, t_l] = \emptyset$. Lastly, for two consecutive intervals $[t_i, t_j]$ and $[t_j, t_k]$ if $ET(e_k) = t_j$ then we assume $e_j \in [t_i, t_j]$.

We introduce the concept of *experience policy* to capture this concept of non-overlapping time intervals. It specifies the totally ordered set of non-overlapping time intervals together with a set of non-negative weights corresponding to each element in the set of time intervals.

Let $\mathcal{P}$ denote the set of all trust-positive events, $\mathcal{Q}$ denote the set of all trust-negative events, and $\mathcal{N}$ denotes all trust-neutral events (that is $\mathcal{E} = \mathcal{P} \cup \mathcal{Q} \cup \mathcal{N}$). We assume that within a given interval all trust-positive events contribute equally to the formation of a trust value and all trust-negative events also do the same. The trust-neutral events contribute nothing. We assign equal numeric weights to all events, trust-positive or trust-negative, within the same given interval. Let v_k^i be the weight of the k^{th} event in the i^{th} interval. We assign a weight of $+1$ if an event is in the set $\mathcal{P}$, -1 if the event is in the set $\mathcal{Q}$, and 0 if the event is in $\mathcal{N}$. Formally, if e_k^i denote the k^{th} event

in the i^{th} interval, then

$$v_k^i = \begin{cases} +1 & \text{, if } e_k^i \in \mathcal{P} \\ -1 & \text{, if } e_k^i \in \mathcal{Q} \\ 0 & \text{, if } e_k^i \in \mathcal{N} \end{cases} \tag{2.1}$$

Definition 11 The *incidents* I_j, corresponding to the j^{th} time interval is the sum of the values of all the events, trust-positive, trust-negative, or neutral for the time interval. If no event happened in j^{th} time interval, then $I_j = \perp$. If n_j is the number of events that occurred in the j^{th} time interval, then

$$I_j = \begin{cases} \perp & \text{, if } \nexists e \in C_E \text{ such that } ET(e) \in [t_{j-1}, t_j] \\ \sum_{k=1}^{n_j} v_k^j & \text{, otherwise} \end{cases} \tag{2.2}$$

Events far back in time does not count as strongly as very recent events for computing trust values. We give more weight to events in recent time intervals than those in distant intervals. To accommodate this in our model, we assign a *non-negative* weight w_i to the i^{th} interval such that $w_i > w_j$ whenever $j < i,\ \ i, j \in \mathbb{N}$. We then define *experience* as follows:

Definition 12 The *experience* of an entity A about another entity B for a particular context c, is the accumulation of all trust-positive, trust-negative, and neutral events that A has with regards to B over a given period of time $[t_0, t_n]$, scaled to be in the range $[-1,1] \cup \{\perp\}$.

Experience has a value in the range $[-1,1] \cup \{\perp\}$. To ensure that the value is within this range we restrict the weight w_i for the i^{th} interval as $w_i = \frac{i}{S} \quad \forall i = 1, 2, \ldots, n,$ where $S = \frac{n(n+1)}{2}$. Then the experience of A with regards to B for a particular context c is given by

$${}_A\mathrm{E}_B^c = \frac{\sum_{i=1}^n w_i I_i}{\sum_{i=1}^n n_i} \tag{2.3}$$

If A does not have any experience with B in the j^{th} interval, then $I_j = \perp$. Thus

$$\begin{aligned} \sum_{i=1}^{n} w_i I_i &= \sum_{i=1}^{j-1} w_i I_i + w_j I_j + \sum_{i=j+1}^{n} w_i I_i \\ &= \sum_{i=1}^{j-1} w_i I_i + \perp + \sum_{i=j+1}^{n} w_i I_i \quad \text{(by property (i) of } \perp) \\ &= \sum_{i=1}^{j-1} w_i I_i + \sum_{i=j+1}^{n} w_i I_i \quad \text{(by property (ii) of } \perp) \end{aligned} \tag{2.4}$$

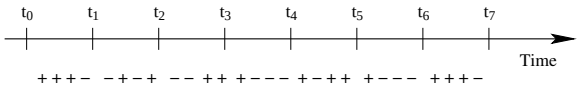
If there is a situation where nothing happened between the truster A and the trustee B over the entire

time period $[t_0, t_n]$, then $I_i = \perp \ \forall\, i = 1, 2, \ldots, n$. As a result, we have $w_i I_i = \perp \ \forall\, i = 1, 2, \ldots, n$ which implies ${}_A\mathrm{E}_B^c = \perp$. The above is different from the situation when ${}_A\mathrm{E}_B^c = 0$. Because, if the number of positive events is equal to number of negative events in each interval, then $I_i = 0, \ \forall\, i = 1, 2, \ldots, n$ and as a result we get ${}_A E_B^c = 0$. But the former case occurs only when there is no interaction between the truster and the trustee over the entire time period.

To illustrate our concept of experience we use the following example. We use the symbol "+" to denote positive events and the symbol "-" to denote negative events.

Example 1

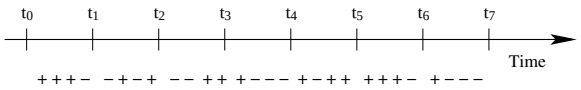
Let us assume that NID_{local} from our example of co-operating network intrusion detection system, monitor the following events related to NID_1 over the time period $\mathrm{t}_0 - \mathrm{t}_7$.



To compute NID_{local}'s experience for NID_1, we divide the time period into the intervals – $[\mathrm{t}_0,\mathrm{t}_1], \ldots [\mathrm{t}_6,\mathrm{t}_7]$. Applying our theory, we have the following incidents: I_0 for interval $[\mathrm{t}_0,\mathrm{t}_1] = +2$, $\mathrm{I}_1 = 0$, $\mathrm{I}_2 = 0$, $\mathrm{I}_3 = -2$, $\mathrm{I}_4 = +2$, $\mathrm{I}_5 = -2$ and $\mathrm{I}_6 = +2$. The weights assigned to each time interval are as follows – w_0 (for interval $[\mathrm{t}_0,\mathrm{t}_1]) = 0.04$, $\mathrm{w}_1 = 0.07$, $\mathrm{w}_2 = 0.11$, $\mathrm{w}_3 = 0.14$, $\mathrm{w}_4 = 0.18$, $\mathrm{w}_5 = 0.21$ and $\mathrm{w}_6 = 0.25$ (for interval $[\mathrm{t}_6,\mathrm{t}_7]$. Thus, the value for NID_{local}'s experience regarding NID_1, over the period $[\mathrm{t}_0,\mathrm{t}_7]$ is 0.00857.

Example 2

Consider the second set of events that NID_{local} monitors over the same time period $\mathrm{t}_0 - \mathrm{t}_7$ for NID_2.



The difference between this set of events and the one in example 1 is that we have more negative events that have happened recently. The total number of trust-positive and trust-negative events are the same in both. We get a value of 0.00286 for experience with this set of events.

2.2.3 Evaluating knowledge

The parameter "knowledge" is more difficult to compute and is, to some extent, subjective. To begin with, each truster must define its own criteria for gradation of knowledge regarding a particular entity. To assign a value to the *knowledge* component, the truster must come up with two values between -1 and +1 for direct knowledge d as well as indirect knowledge or reputation r. How the values are assigned, depends on the scheme and policy of the truster. Also the truster solely

is responsible for assigning the relative weights w_d, w_r for these two types of knowledge, where $w_d, w_r \in [0,1]$ and $w_d + w_r = 1$.

It is possible that the truster has insufficient information to assign a value of d or r. For these types of cases, we assign $\perp$ to those components. If the truster has some numeric value for direct knowledge d, but a '$\perp$' for reputation r, then A evaluates ${}_AK_B^c$ on the basis of direct knowledge only. He applies the same scheme for the other situation. If A does not get any information for both d and r, then ${}_AK_B^c = \perp$. If $d, r \in [-1,1] \cup \{\perp\}$ and $w_d + w_r = 1$ (d and r being the values corresponding to direct and indirect knowledge respectively), then

$$ {}_AK_B^c = \begin{cases} d, \text{if } r = \perp \\ r, \text{if } d = \perp \\ w_d \cdot d + w_r \cdot r, \text{ if } d \neq \perp, r \neq \perp \\ \perp, \text{if } d = r = \perp \end{cases} \tag{2.5} $$

The weights $w_d, w_r \in [0,1]$ are determined by the underlying policy. The value $\perp$ is different from zero. Value 0 implies that after evaluating the information according to its trust policy, the truster's decision is neutral. But the value '$\perp$' implies "lack of information", that is there is not enough data to determine the value of the component. The weights w_d and w_r are specified in terms of a *knowledge policy*.

Example 3

Assume that NID_{local} determines that NID_1 is the same application as itself but running on a different platform. NID_{local} assigns a value of 0.4 for this direct knowledge. It then comes across a piece of information that NID_1's administrator is extremely diligent. By evaluating the *reputation* of NID_1's administrator regarding administration of intrusion detection systems NID_{local} assigns a value 0.65 to reputation. Now NID_{local}'s trust policy guides it to put 70% weight on direct knowledge and 30% weight on reputation. Then, NID_{local} calculates the knowledge component of $(NID_{local} \xrightarrow{IA} NID_1)_t$ as, ${}_{NID_{local}}K_{NID_1}^{IA} = 0.7 \times 0.4 + 0.3 \times 0.65 = 0.475$. By IA we denote the context of correct intrusion alerts.

2.2.4 Evaluating recommendation

An initial recommendation score, V_R, is a value in the range $[-1,1]$ that is provided to the truster by the recommender. To assist the recommender in generating this score, the truster provides a questionnaire to the recommender. The recommender uses the positive values to express his faith in the trustee while uses negative values to express his discontent. If the recommender has no

conclusive decision, he uses zero as recommendation. It is quite possible that the recommender does not return a recommendation sore. In such a case A assigns the value $\perp$ to V_R.

Now a truster A, will, most likely, have a trust relationship with the recommender ψ. The context of this trust relationship will be to act "reliably to provide a service (recommendation, in this case)". This trust relationship will affect the score of the recommendation provided by the recommender. For example, let us say that A trusts ψ to a great extent to provide an appropriate recommendation for B but does not trust ψ' as much as ψ. ψ provides a recommendation score of -0.5 to A and ψ' also provides the same recommendation score. To A, ψ's -0.5 score will have more weight for computing the trust value on B than ψ's, although A will consider both the scores. Scaling the recommendation score based on the trust relationship between the truster and the recommender has one important benefit. Suppose that the recommender tells a lie about the trustee in the recommendation in order to gain an advantage with the truster. If the truster does not trust the recommender to a great degree then the score of this recommendation will be low with the truster. Note also that if the truster distrusts a recommender to properly provide a recommendation, it won't ask for the recommendation to begin with.

We use the trust of the truster on the recommender as a weight to the initial recommendation score returned by the recommender. We had introduced the expression $\mathbf{v}(A \xrightarrow{c} B)_t^N$ earlier to denote the *value* of a normalized trust relationship. This is a value in the range $[-1,1] \cup \{\perp\}$. We use this value as the weight. At this stage we do not specify how we generate this value. We leave that to a later section. Following the above discussion, the *recommendation* ${}_{\psi}R_B^c$ of a recommender ψ for an entity B to the truster A in a context c is given by ${}_{\psi}R_B^c = (\mathbf{v}(A \xrightarrow{rec} \psi)_t^N) \cdot V_R$. In addition, the truster A may get recommendations about the trustee B from many different recommenders. A *recommendation policy* specifies all recommenders and their non-negative weight for a particular trustee. Thus the recommendation value that the truster uses to compute the trust in the trustee is specified as the weighted sum of all recommendation scores scaled to the range $[-1,1] \cup \{\perp\}$. If Ψ is a group of n recommenders then

$$ {}_{\Psi}R_B^c = \begin{cases} \frac{\sum_{j=1}^{n} (\mathbf{v}(A \xrightarrow{rec} j)_t^N) \cdot V_j}{\sum_{j=1}^{n} (\mathbf{v}(A \xrightarrow{rec} j)_t^N)} \end{cases} \tag{2.6} $$

${}_{\Psi}R_B^c = \perp$ is different from ${}_{\Psi}R_B^c = 0$. In the former case, nobody responded and in the latter case all recommenders returned a score '0', that is all of them are neutral about the trustee in the trust context.

Example 4

We continue with our example of NID_{local} trying to establish a trust relationship with other NIDs. Let NID_{local} now ask another NID_3 with whom it already has an established trust relationship

to recommend NID_1 in the context of IA. Let NID_{local} trust NID_3 in the context of "giving recommendation" with 0.8. The recommender NID_3 returns a value (recommendation score) 0.55 for NID_1. Then NID_{local} evaluates the recommendation component of $(NID_{local} \xrightarrow{IA} NID_1)_t$ as ${}_{NID_3}R^{IA}_{NID_1} = 0.8 \times 0.55 = 0.44$.

Next, we consider the case where 4 other NIDs, namely NID_2, NID_3, NID_4, and NID_5 give recommendation about NID_1 in the context IA and their recommendation scores are -0.7, 0.3, 0.8, 0.6 respectively. (NID_2 is NID_1's competitor; so gives a negative recommendation for NID_1). Let NID_{local} trust NID_2, NID_3 and NID_4 with a degrees 0.4, 0.2, 0.75 and 0.5 respectively, in the context of "giving recommendation". (We assume for the time being that these values have been derived somehow from the corresponding trust relationships.) Let $\Psi = \{\text{NID}_2, \text{NID}_3, \text{NID}_4, \text{NID}_5\}$. Then the recommendation is calculated as ${}_{\Psi}R^{IA}_{NID_1} = \frac{0.4\times(-0.7)+0.2\times0.3+0.75\times0.8+0.5\times0.6}{0.4+0.2+0.75+0.5} = 0.368$. Note that NID_2's bias has been offset to a great extent.

2.2.5 Normalizing the trust vector

We mentioned earlier in section 2.2.1 that a truster may give more weight to one of the parameters than others in computing a trust relationship. For example, a truster A may choose to emphasize experience and knowledge more than recommendation. In such case the truster will want to consider the recommendation factor to a lesser extent than experience and knowledge about the trustee. Which particular component needs to be emphasized more than the others, is a matter of trust evaluation policy of the truster. The truster's policy can be trustee specific or can be the same for all trustees. Similarly it can be context specific or context independent. The policy is represented by the truster as a *normalization policy* vector.

Definition 13 The *normalization policy vector* ${}_A\text{W}^c_B$ of a truster A with regards to trustee B in context c is a vector that has the same dimension as the simple-trust vector. The elements are real numbers in the range $[0,1]$ and the sum of all elements is equal to 1.

If the truster has the same normalization policy for all trustees but different for different contexts then we will use the symbol ${}_A\text{W}^c$; for same normalization policy for all context but different trustees we will use the symbol ${}_A\text{W}_B$; finally for same normalization policy for all trustees and for all contexts we will use the symbol ${}_A\text{W}$. Using this normalization policy vector the normalized trust relationship between a truster A and a trustee B at a time t and for a particular context c is given by $(A \xrightarrow{c} B)^N_t = {}_A\text{W}^c_B \odot (A \xrightarrow{c} B)_t$. The $\odot$ operator represents the normalization operator. Let $(A \xrightarrow{c} B)_t = [{}_AE^c_B, {}_AK^c_B, {}_{\Psi}R^c_B]$ be a trust vector such that ${}_AE^c_B, {}_AK^c_B, {}_{\Psi}R^c_B \in [-1,1] \cup \{\perp\}$. Let also ${}_A\text{W}^c_B = [W_E, W_K, W_R]$ be the corresponding trust evaluation policy vector elements such that $W_E +$

$W_K + W_R = 1$ and $W_E, W_K, W_R \in [0,1]$. The $\odot$ operator generates the normalized trust relationship as: $(A \xrightarrow{c} B)_t^N = {}_A\mathrm{W}_B^c \odot (A \xrightarrow{c} B)_t = [W_E, W_K, W_R] \odot [{}_AE_B^c, {}_AK_B^c, {}_\psi R_B^c] = [W_E \cdot {}_AE_B^c, W_K \cdot {}_AE_B^c, W_R \cdot {}_\psi R_B^c] = [{}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c]$. It follows from above that each element ${}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c$ of the normalized trust vector also lies within $[-1,1] \cup \{\perp\}$.

Example 5

Continuing with the example, the simple trust relationship between NID_{local} and NID_1 at time t is specified as, $(NID_{local} \xrightarrow{IA} NID_1)_t = [0.00857, 0.475, 0.368]$. NID_{local} decides to put 60% weight on experience, 30% on knowledge and rest 10% on recommendation. Then NID_{local}'s trust evaluation policy vector is, ${}_{NID_{local}}\mathbf{W} = [0.6, 0.3, 0.1]$. Hence the normalized trust vector $(NID_{local} \xrightarrow{IA} NID_1)_t^N$ is $[0.6, 0.3, 0.1] \odot [0.00857, 0.475, 0.368] = [0.005142, 0.1425, 0.0368]$.

The normalization policy together with the experience policy and the knowledge policy form the *trust evaluation policy* of the truster.

2.2.6 Value of the normalized trust vector

So far we have defined a trust relationship in terms of a vector which is *normalized* by a trust policy. Recall from section 2.2.4 that there is at least one scenario in which we need to use a trust value as a weight for a real number, namely for computing recommendations. Thus it seems appropriate to define the concept of a *value* corresponding to the normalized trust vector.

Definition 14 The *value* of a normalized trust relationship $(A \xrightarrow{c} B)_t^N = [{}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c]$ is a number in the range $[-1,1] \cup \{\perp\}$ and is defined as $\mathbf{v}(A \xrightarrow{c} B)_t^N = {}_A\hat{E}_B^c + {}_A\hat{K}_B^c + {}_\psi\hat{R}_B^c$.

The value for a trust relationship allows us to revise the terms "trust" and "distrust" as follows: 1. If the value, T, of a normalized trust relationship is such that $0 < T \leq 1$ then it is trust. 2. If the value, T, of a normalized trust relationship is such that $-1 \leq T < 0$ then it is distrust. 3. If the value, T, is 0 then it is neither trust nor distrust. 4. If the value, T, is $\perp$ then it is *undefined*.

Example 6

With our running example, the value of the normalized trust between NID_{local} and NID_1 in the context IA at time t is given as, $\mathbf{v}(NID_{local} \xrightarrow{IA} NID_1)_t^N = 0.005142 + 0.1425 + 0.0368 = 0.1844$. Since the value lies within the range $(0,1]$, the "level of trust" of NID_{local} with NID_1 in the context IA at time t is 0.1844.

2.2.7 Trust dynamics

Trust (and distrust) changes over time. Let us assume that we have initially computed a trust relationship $\vec{T}_{t_i}$ at time t_i, based on the values of the underlying parameters at that time. Suppose now that we try to recompute the trust relationship $\vec{T}_{t_n}$ at time t_n. We claim that even if the underlying parameters do not change between times t_i and t_n, the trust relationship will change. To model *trust dynamics* (the change of trust over time) we borrow from observations in the social sciences that indicate that human abilities and skills respond positively to practice, in a learning-by-doing manner, and negatively to non-practice [6]. We observe that the general tendency is to forget about past happenings. This leads us to argue that trust (and distrust) tends towards neutrality as time increases. Initially, the value does not change much; after a certain period the change is more rapid; finally the change becomes more stable as the value approaches the neutral (value = 0) level. We assert that $\lim_{t\to\infty} \mathbf{v}(\vec{T}_t) = 0$. Thus trust dynamics can be represented by the graph shown in figure 2.1. How fast trust (or distrust) will decay over time, is, we propose, dependent on the truster's

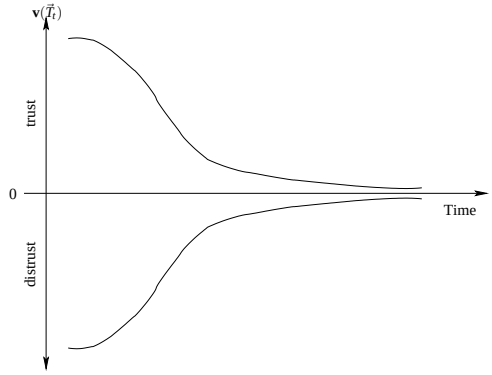


Figure 2.1: Graph Showing the Nature of Trust Dynamics

policy. The truster may choose to forget about trust relationships which are 3 years old or 5 years old. The model cannot dictate this. Our goal is to provide a basis by which the truster can at least estimate, based on the truster's individual perception about this, the trust at time t_n. We further believe that trust relationship at present time is not only dependent on the values of the underlying parameters, but also on the "decayed" value of the previous trust. We discuss this in more details in the next section.

Let $\mathbf{v}(\vec{T}_{t_i})$, be the value of a trust relationship, $\vec{T}_{t_i}$, at time t_i and $\mathbf{v}(\vec{T}_{t_n})$ be the decayed value of the same at time t_n. Then the *time-dependent value* of $\vec{T}_{t_i}$ is defined as follows:

Definition 15 The *time-dependent value* of a trust relationship $\vec{T}_{t_i}$ from time t_i, computed at present time t_n, is given by $\mathbf{v}(\vec{T}_{t_n}) = \mathbf{v}(\vec{T}_{t_i})e^{-(\mathbf{v}(\vec{T}_{t_i})\Delta t)^{2k}}$, where $\Delta t = t_n - t_i$ and k is any small integer ≥ 1.

The value k determines the rate of change of trust with time and is assigned by the truster based on its perception about the change. If $\Delta t = 0$ that is at $t_n = t_i$, $e^{-(\mathbf{v}(\vec{T}_{t_i})\Delta t)^{2k}} = 1$ and hence $T_n = T_i$. When $\Delta t \to \infty$, then $e^{-(T_i \Delta t)^{2k}} \to 0$ and hence $T_n \to 0$. This corroborates the fact the time-dependent value of the last known trust value is asymptotic to zero at infinite time. The parameter k is specified by the truster's *dynamic policy* regarding the trustee in the specific context.

To obtain the trust vector $\vec{T}_{t_n}$ at time t_n, we distribute the value $\mathbf{v}(\vec{T}_{t_n})$ evenly over the components. The rational behind this is that between t_i and t_n we do not have sufficient information to assign different weights to the different components. Thus we have the time-dependent vector as $\vec{T}_{t_n} = [\frac{\mathbf{v}(\vec{T}_{t_n})}{3}, \frac{\mathbf{v}(\vec{T}_{t_n})}{3}, \frac{\mathbf{v}(\vec{T}_{t_n})}{3}]$.

2.2.8 Trust vector at present time

The trust of a truster A on a trustee B in a context c at time t_n depends not only on the underlying components of the trust vector but also on the trust established earlier at time t_i. Consider for example that at time t_i NID_{local} trusts NID_1 to the fullest extent (value = 1). At time t_n NID_{local} re-evaluates the trust relationship and determines the value to be -0.5 (distrust). However, we believe that NID_{local} will lay some importance to the previous trust value and will not distrust NID_1 as much as a -0.5 value. So, the normalized trust vector at t_n is a linear combination of time-dependent trust vector and the normalized trust vector calculated at present time. The weight NID_{local} will give to old trust vector and present normalized trust vector is, again, a matter of policy. This is specified as the *history_weight policy* of the truster which consists of two values, α and β corresponding to the present normalized trust vector and the time-dependent vector. This leads us to refine the expression for normalized trust vector at time t_n as follows. Let $\hat{T}$ be the time-dependent trust vector derived from $(\vec{T}_{t_i})$ at time t_n.

Definition 16 The normalized trust relationship between a truster A and a trustee B at time t_n in a particular context c is given by

$$(A \xrightarrow{c} B)^N_{t_n} = \begin{cases} [{}_A\hat{E}^c_B, {}_A\hat{K}^c_B, {}_\psi\hat{R}^c_B] & \text{if } t_n = 0 \\ [\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}] & \text{if } t_n \neq 0 \text{ and } {}_A\hat{E}^c_B = {}_A\hat{K}^c_B = {}_\psi\hat{R}^c_B = 0 \\ \alpha \cdot [{}_A\hat{E}^c_B, {}_A\hat{K}^c_B, {}_\psi\hat{R}^c_B] + \beta \cdot [\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}] & \\ \qquad \text{if } t_n \neq 0 \text{ and at least one of } {}_A\hat{E}^c_B, {}_A\hat{K}^c_B, {}_\psi\hat{R}^c_B \neq 0 & \end{cases} \tag{2.7}$$

where $\alpha \cdot [{}_A\hat{E}^c_B, {}_A\hat{K}^c_B, {}_\psi\hat{R}^c_B] + \beta \cdot [\frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}, \frac{\mathbf{v}(\hat{T})}{3}] = [\alpha \cdot {}_A\hat{E}^c_B + \beta \cdot \frac{\mathbf{v}(\hat{T})}{3}, \alpha \cdot {}_A\hat{K}^c_B + \beta \cdot \frac{\mathbf{v}(\hat{T})}{3}, \alpha \cdot {}_\psi\hat{R}^c_B + \beta \cdot \frac{\mathbf{v}(\hat{T})}{3}]$, $\alpha, \beta \in [0,1]$ and $\alpha + \beta = 1$.

2.3 Comparison Operation on Trust Vectors

We are now in a position to determine the relative trustworthiness of two trustees. The need for such comparison occurs in many real life scenarios. Consider the following example. Suppose entity A gets two conflicting pieces of information from two different sources B and C. In this case A will probably want to compare its trust relationships with entities B and C and accept the information that originated from the "more" trustworthy entity. This motivates us to define a comparison operator on trust relationships.

Let T and T' be two normalized trust relationships at time t. We introduce the following notion of compatibility between two trust relationships.

Definition 17 Two trust relationships, T and T' are said to be *compatible* if the trust relationships have been defined under the same trust evaluation policy vector, the trust relationships are at the same time instances, and the context $c(T)$ for the trust relationship T is the same as the context $c(T')$ for T', that is $c(T) = c(T')$. Otherwise the two trust relationships are said to be *incompatible*.

Note that the definition of compatibility between two trust relationships does not explicitly involve information about the trusters or trustees. However, since the trust relationships are being compared under the same trust evaluation policy vector they must involve the same truster. The most intuitive way to compare two trust relationships T and T' is to compare the values of the trust relationships in a numerical manner. Thus for A to determine the relative levels of trustworthiness of B and C, A evaluates $\mathrm{v}(A \xrightarrow{c} B)_t^N$ and $\mathrm{v}(A \xrightarrow{c} C)_t^N$. If $\mathrm{v}(A \xrightarrow{c} B)_t^N > \mathrm{v}(A \xrightarrow{c} C)_t^N$, then A trusts B more than C in the context c. We say that T *dominates* T', given by $T \succ T'$. However, if $\mathrm{v}(A \xrightarrow{c} B)_t^N = \mathrm{v}(A \xrightarrow{c} C)_t^N$, A cannot judge the relative trustworthiness of B and C. This is because there can be two vectors whose individual component values are different but their scalar values are the same. For such cases we need to compare the individual elements of the two trust relationships to determine the relative degree of trustworthiness. In addition, for the same reasons, it is better to determine relative trustworthiness of B and C on the basis of component values rather than breaking the tie arbitrarily.

Let $(A \xrightarrow{c} B)_t^N = [{}_A\hat{E}_B^c, {}_A\hat{K}_B^c, {}_\psi\hat{R}_B^c]$ and $(A \xrightarrow{c} C)_t^N = [{}_A\hat{E}_C^c, {}_A\hat{K}_C^c, {}_\psi\hat{R}_C^c]$ such that $\mathrm{v}(A \xrightarrow{c} B)_t^N = \mathrm{v}(A \xrightarrow{c} C)_t^N$. Let also the underlying trust evaluation policy vector be given by ${}_AW = (w_1, w_2, w_3)$ where $w_1 + w_2 + w_3 = 1$ and $w_i \in [0,1]\ \forall i = 1,2,3$. To determine the dominance relation between T and T' we first determine the *ordered* trust relationships $\bar{T}$ and $\bar{T'}$ corresponding to T and T'.

Definition 18 The *ordered* trust relationship $\bar{T}$ is generated from a normalized trust relationship T as follows:

1. Order the w_i's in the trust evaluation policy vector corresponding to T in descending order of magnitude.
2. Sort the components of the trust vector T according to the corresponding weight components.

We compare the two ordered trust relationships $\overline{T}$ and $\overline{T'}$, corresponding to T and T', component-wise to determine the dominance relation between the two. Note that we assume that the same underlying trust evaluation policy vector has been used to determine the trust relationships. If the first component of $\overline{T}$ is numerically greater than the first component of $\overline{T'}$ then $T \succ T'$. Else if the first components are equal then compare the second components. If the second component of $\overline{T}$ is greater than the second component of $\overline{T'}$ then $T \succ T'$, and so on. If weights are equal for first two (or, all three) components in the ordered trust relationships, then $T \succ T'$ only when both components (or, all three components) of T are numerically greater than those of T'. In the comparison process we assume that the value $\perp$ is dominated by all real numbers. If we cannot conclude a dominance relation between the two trust relationship, then we say that the two trust relationships are *incomparable*. This is formalized by the following definition.

Definition 19 Let T and T' be two trust relationships and $\overline{T}$ and $\overline{T'}$ be the corresponding ordered trust relationships. Let also $\overline{T_i}$ and $\overline{T_i'}$ represent the ith component of each ordered trust relationships and w_i represent the i^{th} weight component in the corresponding trust evaluation policy vector. T is said to dominate T' if any one of the following holds:

1. $\mathrm{v}(T) > \mathrm{v}(T')$; or
2. if $\forall\ i, j,\ i \neq j,\ (w_i = w_j)$ then $\forall\ i,\ \overline{T_i} > \overline{T_i'}$; or
3. if $\exists\ i, \overline{T_i} > \overline{T_i'}$ and for $k = 1 \ldots (i-1),\ \overline{T_{i-k}} \not< \overline{T_{i-k}'}$

Otherwise T is said to be *incomparable* with T'.

2.4 Combining Trust Relationships

We have defined a basic trust relationship as a binary relation between two different entities – a truster and a trustee. However, today's world of information exchange involves many cooperative entities in a relationship within a specified context. Combination of trust is needed for the interoperability of these cooperating agents. Whenever a group of agents are working together, combination of their individual trust relationship is necessary to have an idea about the expected behavior of the group. Keeping this in mind we now formalize combination operators for trust relationships. Different possibilities like one-to-many, many-to-one, and many-to-many relationships are addressed. We also formalize the effect of reconfiguration of these groups on the corresponding

trust relationships.

2.4.1 Trust relationship between a truster and a group of trustees

In real life, we often encounter situations where we have to take decisions based on information coming from different sources. Consider the scenario where an entity has existing trust relationships with different service providers for a particular service. The truster expects some service which is provided collectively by the service providers. The truster has some expectation from each individual provider. To have an idea about the service provided by the group, the combined trust of the service providers needs to be estimated. Therefore, the receiver needs a mechanism to combine the existing trust relationships to estimate an initial *composite trust relationship*. The group of service providers is considered as a single entity (trustee). Once the combination is done, the truster no longer considers the trust relationships with individual trustee. The truster begins with the combined group as a single entity and subsequently a trust relationship with the group evolves.

Let an entity A have trust relationships with two different entities B and C in the same context c at time t. A decides to have a trust relationship with the combined group BC in the same context. It is clear that individual trust relationships of both B and C will have effect on the resulting trust vector. However, the individual trust relationships will have different degrees of effect. This is represented by A putting different weights on the trustees to evaluate their relative importance in the trustee group. Once the group is formed the trust $(A \xrightarrow{c} BC)_t^N$ evolves as a new trust relationship. Thus we define the *initial* trust relationship between A and BC in context c as follows.

Definition 20 Let at time t a truster A have two trust relationships, $(A \xrightarrow{c} B)_t^N$ and $(A \xrightarrow{c} C)_t^N$ with trustees B and C respectively. If $\oplus$ is the *trust combination operator* then the trust relationship between A and the group BC is defined as $(A \xrightarrow{c} BC)_t^N = (A \xrightarrow{c} B)_t^N \oplus (A \xrightarrow{c} C)_t^N$.

A *trustee group policy* specifies the non-negative weights of experience, knowledge and recommendation for all individual trustees within the group. The trust combination operator is specified as follows. For each component of the combined trust vector $(A \xrightarrow{c} BC)_t^N$, the operator $\oplus$ takes a weighted sum of respective components of $(A \xrightarrow{c} B)_t^N$ and $(A \xrightarrow{c} C)_t^N$. The sum of the weights for each component equals 1. Thus, if $(A \xrightarrow{c} B)_t^N = ({}_A\hat{E}_B, {}_A\hat{K}_B, {}_{\psi_B}\hat{R}_B), (A \xrightarrow{c} C)_t^N = ({}_A\hat{E}_C, {}_A\hat{K}_C, {}_{\psi_C}\hat{R}_C)$ be the trust vectors, then the combined vector is given by $(A \xrightarrow{c} BC)_t^N =$

$({}_A\hat{E}_{BC}, {}_A\hat{K}_{BC}, {}_{\psi_{BC}}\hat{R}_{BC})$

$$
\begin{aligned}
{}_A\hat{E}_{BC} &= w^E_1 {}_A\hat{E}_B + w^E_2 {}_A\hat{E}_C && (2.8)\\
{}_A\hat{K}_{BC} &= w^K_1 {}_A\hat{K}_B + w^K_2 {}_A\hat{K}_C && (2.9)\\
{}_{\psi_{BC}}\hat{R}_{BC} &= w^R_1 {}_{\psi_B}\hat{R}_B + w^R_2 {}_{\psi_C}\hat{R}_C && (2.10)
\end{aligned}
$$

Here, $w^E_1 + w^E_2 = 1$, $w^K_1 + w^K_2 = 1$ and $w^R_1 + w^R_2 = 1$. The weights w^{comp}_i is weight assigned to i^{th} trustee for the component $comp \in \{E, K, R\}$ and $w^{comp}_i \in [0,1] \forall i,\ \forall comp$.

Note that, A has two groups of recommenders ψ_B and ψ_C for B and C respectively. There are five relations possible for these two groups, namely 1. $\psi_B = \psi_C$ 2. $\psi_B \cap \psi_C = \emptyset$ 3. $\psi_B \subset \psi_C$ 4. $\psi_B \supset \psi_C$ 5. $\psi_B \cap \psi_C \neq \emptyset$ and none of 1, 3, 4 hold. The truster A forms a new list of recommender ψ_{BC} for the combined group BC where, $\psi_{BC} = \psi_B \cup \psi_C$, irrespective of the above five relations between ψ_B and ψ_C. If the truster has m trust relationships with trustees $B_1, B_2, \ldots, B_m$, we can easily generalize the above concept for the group of trustees $\mathcal{G} = \{B_1, B_2, \ldots, B_m\}$ as $(A \xrightarrow{c} \mathcal{G})^N_t = (A \xrightarrow{c} B_1)^N_t \oplus (A \xrightarrow{c} B_2)^N_t \ldots \oplus (A \xrightarrow{c} B_m)^N_t$. The operator $\oplus$ takes the weighted sum of the corresponding components of the vectors.

2.4.2 Trust relationship between a group of trusters and a single trustee

Next, we address the situation where different trusters having different trust relationships with a particular entity in a context, form a group. After forming a group the trusters work as a single truster entity. We need to define a way to combine these different trust relationships to get the initial trust for the group. This initial trust gives the starting point of a trust relationship between two entities (a group and a single trustee). Thereafter, this trust evolves as before. But before grouping, different trusters have their own policy to evaluate the trustee for the same context. In other words, though trust context is same, there are different trust policies. Unless all the trusters agree to a common policy, as well as a common criteria for evaluation, there can not be a single trust relationship. To achieve this, there should be a *consensus* among the members.

At this stage we need to discuss some issues. Let an entity A and B have trust vectors about an entity C in some context c at time t. Now A and B want to collaborate and work as a single truster. The initial trust for the group in the context c is derived from their individual relationship with C. Let A have more experience and knowledge than B in terms of trust relationship with C in some context c. But B has stronger recommendations about C. Therefore, for initial group trust, for experience and knowledge, A will play the major role determining those. The recommendation component of the initial trust of the group will have more influence of recommendation value of

B's trust relationship with C. So, for each component of the trust vector, the group of trusters has an ordering according to their individual contribution for the component. For each component, they have to assign weights to each individual truster in the group according to their relative ordering. How the weights would be assigned is determined by the consensus the group arrives at during the time of group formation. After that, the components are evaluated for the whole group as a single entity.

Trust consensus for a group of trusters

Definition 21 The *trust-consensus* of a group of trusters is defined as the agreement among all members to build a common basis for evaluating a combined trust relationship.

Let $A_1, A_2, \ldots, A_m$ be m trusters trying to form a group say $\mathcal{G}$, to build a single trust relationship with a trustee B in some common context c. All these trusters have different trust relationships with B in context c at the present time t. So there are m existing trust relationships $(A_1 \xrightarrow{c} B)_t^N, (A_2 \xrightarrow{c} B)_t^N, \ldots, (A_m \xrightarrow{c} B)_t^N$ at t. The objective is to get a trust relationship $(\mathcal{G} \xrightarrow{c} B)_t^N$ where $\mathcal{G} = \{A_1, A_2, \ldots, A_m\}$.

The members need to agree to the following things before formation of the group: (i) For each component, a set of weights to assign relative importance of the members. (ii) A common trust evaluation policy vector to assign weights to each component of combined trust. (iii) A common interval length to determine experience, as well as trust. (iv) Common weights for direct knowledge and reputation in the group trust. (v) A common set of recommenders whom the group consider for providing recommendation about the trustee. (vi) A common policy to evaluate trust relationship with a recommender.

For the 5^{th} point above, let $\psi_1, \ldots, \psi_m$ be m group of recommenders who have provided recommendation for B in context c to the truster $A_1, \ldots, A_m$ respectively. Now the group $\mathcal{G}$ of trusters forms a new group Ψ of recommenders, where $\Psi = \bigcup_{i=1}^{m} \psi_i$.

Let $|\Psi| = k$ (i.e., there are k distinct recommenders in the group) Ψ. Each of the A_i's may not have a trust relationship with all of these k recommenders (when $\psi_i \cap \psi_j = \emptyset$, $\forall i \neq j$). Therefore, the group $\mathcal{G}$ evaluates trust relationship according to their newly formed policy for establishing

trust with a recommender. Therefore, we have $(\mathcal{G} \xrightarrow{c} B)_t^N = [{}_{\mathcal{G}}\hat{E}_B^c, {}_{\mathcal{G}}\hat{K}_B^c, {}_{\Psi}\hat{R}_B^c]$ where,

$$ {}_{\mathcal{G}}\hat{E}_B^c = \sum_{i=1}^{m} w_i^E \cdot_{A_i} \hat{E}_B^c \tag{2.11} $$

$$ {}_{\mathcal{G}}\hat{K}_B^c = \sum_{i=1}^{m} w_i^K \cdot_{A_i} \hat{K}_B^c \tag{2.12} $$

$$ {}_{\Psi}\hat{R}_B^c = \sum_{i=1}^{m} w_i^R \cdot_{A_i} \hat{R}_B^c \tag{2.13} $$

Here, $w_i^{comp} \in [0,1]$ and $\sum_{i=1}^{m} w_i^{comp} = 1, \forall comp \in \{E,K,R\}$.

After arriving at a *trust-consensus*, group $\mathcal{G}$ works as a single entity to work further with the trustee B according to their trust-consensus.

2.4.3 Trust relationship between a group of trusters and a group of trustees

We now explore the situation when a group of trusters $\mathcal{G}_r$ forms a trust relationship with a group of trustees $\mathcal{G}_e$ in some common context c. Though this is a complicated concept, we can formalize this by combining the above two cases. Combination can take place in different ways.

1. If the group of trustees $\mathcal{G}_e$ already exists, then each truster A_i must already have, or must build a trust relationship $(A_i \xrightarrow{c} \mathcal{G}_e)_t^N$ as described in section 2.4.1. Then A_i's form the truster group $\mathcal{G}_r$ with $\mathcal{G}_e$, considering $\mathcal{G}_e$ as a single trustee, as described in section 2.4.2.
2. If the truster group $\mathcal{G}_r$ already exists with m different trust relationships like $(\mathcal{G}_r \xrightarrow{c} B_i)_t^N$ for $i = 1,2,\ldots,m$, then $(\mathcal{G}_r \xrightarrow{c} \mathcal{G}_e)_t^N$ can be formed as in section 2.4.1.
3. If neither group of trusters or trustees exist, either of the group has to be formed first and then the other group is formed as explained above.

We have defined combination operations for one truster-many trustees, many trusters-one trustee and many trusters-many trustees. The group is formed under a common trust policy. Next we examine the effect of reconfiguration of a group on the trust relationship.

2.4.4 Reconfiguration of a group

After the group is formed, some member may leave, or some new member may join the group. This contraction (or, expansion) of the group can happen in steps or, in one instance. That is, old members can leave one by one or, together. Similarly, new members can join in subsequent time instances, or as a whole group. We now address the issue of reconfiguration of group of trusters (or, trustees) over time, and examine its effect on the existing trust relationship.

Reconfiguration of a trustee group

Let at time t_n, there be a trust relationship between a truster A and a group of trustees $\mathcal{G}$, where $\mathcal{G} = \{B_1, B_2, \ldots, B_m\}$. Therefore, at t_n, we have the trust relationship $(A \xrightarrow{c} \mathcal{G})^N_{t_n}$. Now, let at t_{n+1}, a new trustee B_{m+1} join the group $\mathcal{G}$. Then to build the new trust relationship (rather, we say to "reconfigure" the existing trust relationship) $(A \xrightarrow{c} \mathcal{G}')^N_{t_{n+1}}$ where $\mathcal{G}' = \mathcal{G} \cup \{B_{m+1}\}$, A reassigns the weights for each component for $\mathcal{G}$ and B_{m+1}, according to the trust policy without violating the existing conditions. A does not combine B_{m+1} with existing m trustees, rather he combines two entities $\mathcal{G}$ and B_{m+1}, treating $\mathcal{G}$ as a single entity. That is, in case of $(A \xrightarrow{c} \mathcal{G}')^N_{t_{n+1}}$, the weights $w^{comp}_{\mathcal{G}}, w^{comp}_{B_{m+1}} \in [0,1]$ and $w^{comp}_{\mathcal{G}} + w^{comp}_{B_{m+1}} = 1$ where, comp $\in \{E, K, R\}$. The truster A also needs to update the recommender list by adding the recommenders involved in the trust relationship $(A \xrightarrow{c} B_{m+1})^N_{t_{n+1}}$.

We now consider the situation where at t_{n+1}, instead of joining the group, some trustee B_i leaves the group. That is, $\mathcal{G}' = \mathcal{G} - \{B_i\}$ for some $i \in \{1, 2, \ldots, m\}$. Then the truster needs not to change the policy to evaluate the trust relationship. Because, after the trustee group is formed, truster considers the trustee group as a single entity. This trust relationship has evolved over time and removal of a trustee does not change the truster's policy of trust evaluation. The trust relationship will evolve further with the reduced trustee group. Impact of absence of a trustee on the trust relationship is noticed accordingly.

The above ideas can easily be extended if a group of new trustees (i.e., more than one new trustee) join (or, leave) the existing group of trustees at a time. If a new group of trusters $\mathcal{G}''$ joins, then $\mathcal{G}' = \mathcal{G} \cup \mathcal{G}''$ where $\mathcal{G}'' = \{B_{m+1}, \ldots, B_n\}$ and $n > m$. The recommender's list is also updated accordingly. If a subgroup $\mathcal{G}''$ of trustees leaves the group $\mathcal{G}$ ($\mathcal{G}'' \subset \mathcal{G}$), then $\mathcal{G}' = \mathcal{G} - \mathcal{G}''$.

Reconfiguration of a truster group

Let at time t_n, there be a trust relationship between a group of trusters $\mathcal{G}$ and a trustee B, where $\mathcal{G} = \{A_1, A_2, \ldots, A_m\}$. Therefore, at t_n, we have the trust relationship $(\mathcal{G} \xrightarrow{c} B)^N_{t_n}$. Now, let at t_{n+1}, a new truster A_{m+1} joins the group $\mathcal{G}$. The new group $\mathcal{G}' = \mathcal{G} \cup \{A_{m+1}\}$ builds a new trust relationship $(\mathcal{G}' \xrightarrow{c} B)^N_{t_{n+1}}$ with B. Since $\mathcal{G}$ is a group that has been formed earlier, at t_{n+1} we no longer consider individual component values for all truster. The reason is, whenever a group is formed, at the time of formation, the members are ranked according to their relative importance for each component. After formation each member works in the same way as other with the trustee. So, after formation there is no discrimination among the existing group members. A trust-consensus is made between the two truster entities, $\mathcal{G}$ and A_{m+1}. The component values of the new truster A_{m+1} is, therefore compared to the corresponding component values of the group $\mathcal{G}$. A_{m+1}

may be a newcomer in the field (with less experience and knowledge) or may be senior enough to get more importance than the formed group $\mathcal{G}$, when he is about to join $\mathcal{G}$. In the latter case, in $(\mathcal{G}' \xrightarrow{c} B)^N_{t_{n+1}}, A_{m+1}$ will have higher weights for the components in which he dominates.The agreement between the joining member and the group determines the relative importance of the two entities in that trust relationship.

This idea is easily extended to the situation where at t_{n+1} more than one new truster join $\mathcal{G}$. In that case, $\mathcal{G}' = \mathcal{G} \cup \mathcal{G}''$ where $\mathcal{G}'' = \{A_{m+1}, \ldots, A_n\}$ and $n > m$. In this case ordering is done for each of the component values of $\mathcal{G}$, and $A_{m+1}, \ldots, A_n$. The trust-consensus is made accordingly.

Removal of a truster from the group does not affect the group trust-consensus. The group continues its trust relationship with the trustee as earlier. The trust evolves over time as before. Absence of a member does not affect the trust relationship as long as the trustee group remains unaltered. It is also true if more than one member leave the group at a time. Changing of group policy, or arriving at a new trust-consensus depends totally on the group. If a significant number of members have left the consortium, the existing members may go for a revised agreement on how to evaluate the trust relationship thereafter. Suppose at time t_n we have a trust relationship $(\mathcal{G} \xrightarrow{c} B)^N_{t_n}$. Let us assume that at each subsequent time intervals one or more members leave the group. Then there is a sequence of time $\{t_k\}$ after the time t_n such that at t_{n+k} only one member from the group remains in $\mathcal{G}$. Then at t_{n+k}, the solitary member can go with the existing trust policy and scheme to evaluate trust, or he can establish a new one-to-one trust relationship with the trustee.

Chapter 3

The VTrust Trust Management System

The "Vector" trust model was presented in the last chapter. To use the model we need a corresponding trust management system. A trust management system is a comprehensive framework designed to facilitate the specification, analysis and management of trust relationships. It focuses on specifying and interpreting security policies, credentials, and relationships. The trust management system also provides trust establishment, trust evaluation, trust monitoring and trust analysis services. These require, among others, a language to specify trust relationships and a mechanism to store and manipulate the same. In this chapter, we describe the trust management system that we have developed to accompany the new trust model. We call the system VTrust (from Vector Trust model) trust management system. Major components of the trust management system include a database engine to store and manage trust data, a trust specification engine for defining and managing trust relationships, a trust analysis engine to process results of a trust query, a trust evaluation engine for evaluating trust expressions and a trust monitor for updating trust relationship information in the database engine. We have also developed an SQL like language, called TrustQL, to interact with the trust management system.

3.1 The VTrust System Architecture

The high level system architecture consists of the components as shown in the following figure 3.1.

Values of the different parameters needed for the computation of trust relationships are maintained in the VTrust database. The truster interacts with the trust management system through the external interface. The communication is done using the language TrustQL that we have developed. The TrustQL language parser in the interface parses the command and sends it to the appropriate component in the next layer. This layer has the following major components. A *speci-*

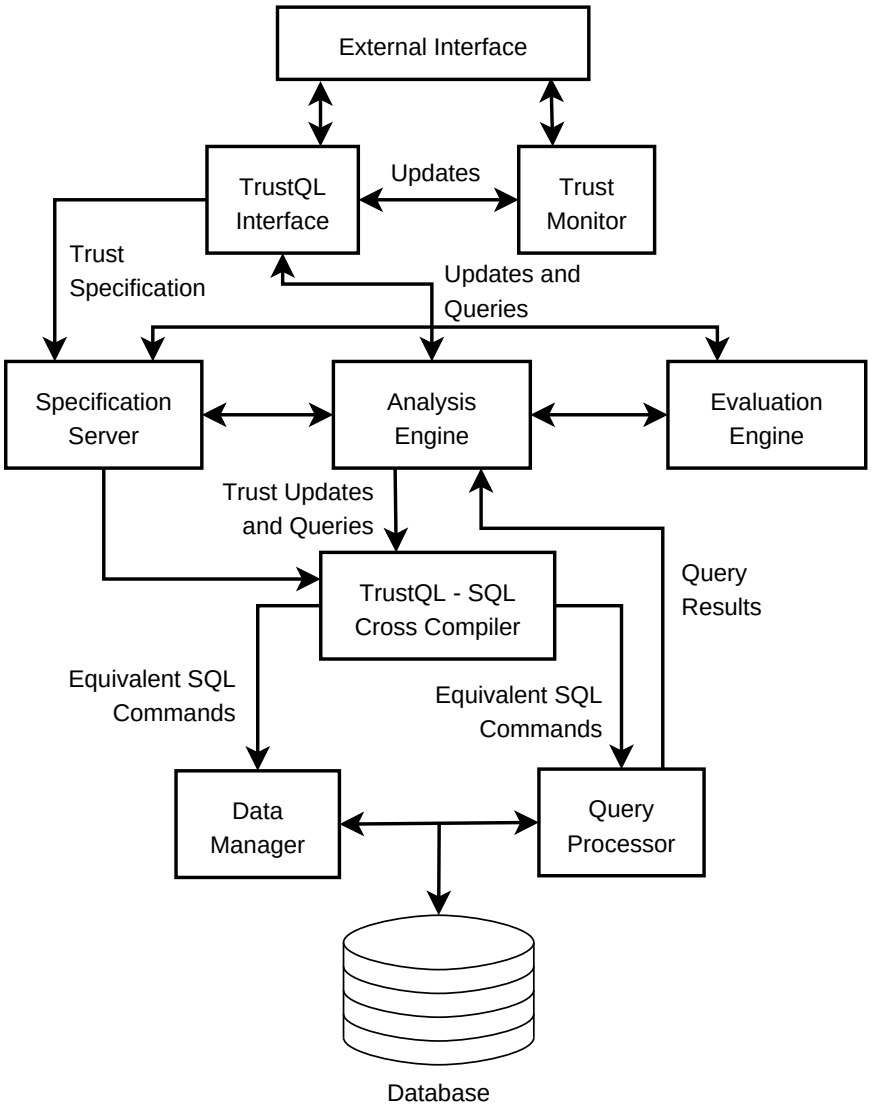


Figure 3.1: Trust Management System Architecture

fication server is managing and updating the trust database schema. The *analysis engine* processes all trust related queries. It interacts with specification server and an *evaluation engine*. The latter is responsible for computing trust related information according to the underlying model. The evaluation engine takes a parsed trust query string, finds the associated information and policy, and returns the final trust vector and value to the analysis engine. The *trust monitor* is responsible for acquiring relevant trust formulation parameters. It maintains the VTrust database, updates the trust data while truster and trustee interacts with each other and also updates periodically trust component values like experience and knowledge.

All these information (trusters, trustees, recommenders, policies and trust parameter information) are stored in the VTrust database. The database is implemented as described in section 3.2.

Since TrustQL can not interact with the database directly, an SQL translator beneath the component layer does this job. The specification server, analysis engine and evaluation engine takes a trust operation specified in TrustQL and maps the command to an equivalent SQL command to interact with the underlying database. After receiving an answer from the database, each of those components again does a reverse mapping to output the answer in terms of TrustQL.

The following algorithm is used by a truster to compute the trust relationship with a trustee for a given context at any given time.

Algorithm 1

1. If not already available, initialize the truster's trust evaluation policy corresponding to the trustee and the specific context. If needed update the same to reflect current circumstances.

2. Initialize dynamic policy and history_weight policy if not already available. Update as needed.

3. Compute truster's experience with trustee.

   (a) Determine last point in time when trust was evaluated for current trustee in the given context. If such a time exists call it t_{last}.

   (b) Read off experience values from database starting from most recent first till either t_{last} or start of experience table.

   (c) Apply experience policy to evaluate current experience value.

4. Compute truster's knowledge with trustee by applying knowledge policy to current direct knowledge and reputation values.

5. Compute recommendation value for trustee.

6. Compute truster's simple trust on trustee using values obtained in steps 3 - 5. Apply normalization policy as appropriate to the simple trust.

7. Let trust value at t_{last} be termed T_{last} (assuming available); compute decayed value for T_{last} by applying dynamic policy to it.

8. Combine trust values obtained in steps 6 and 7 using the history_weight policy to get the truster's current trust relationship with the trustee in the given context.

9. Record current time of trust evaluation as t_{last} corresponding to this truster, trustee and context.

3.2 Conceptual Trust Model

Conceptually, we can model the underlying trust components using an Entity-Relationship model. The relational entities include ENTITY, TRUST, CONTEXT, EVENTS, EXPERIENCE, KNOWLEDGE, RECOMMENDATION, EXP_POLICY, KNOW_POLICY, NORMAL_POLICY, DYN_POLICY and HIST_WT_POLICY. All these entities are represented as table in a relational database with columns representing the 'attributes' of those relational entities.

In subsequent sections, we discuss the relationship between the entities in the model. We will represent a relational entity and its corresponding table in CAPS and the attributes will be represented in *italics*.

3.2.1 Inter-relationship of relational entities

ENTITY includes Truster, Trustee, or Recommender as *role*. A TRUST involves a *Truster*, a *Trustee*, a *Context*, and is evaluated on a particular *Date*. TRUST depends on the following entities and relationships: EXPERIENCE, KNOWLEDGE, RECOMMENDATION, NORMAL_POLICY, DYN_POLICY and HIST_WT_POLICY. All these involve the same *Truster, Trustee, Context, Date* as in TRUST.

EXPERIENCE depends on EVENTS and EXP_POLICY and returns a *Evalue*. The entity EVENTS is a log of events happened between the truster and the trustee in the context during a certain interval of time. EXP_POLICY specifies the length of time interval. KNOWLEDGE returns *Kvalue* which is evaluated based on KNOW_POLICY which determines weights for direct knowledge (*Dknol_wt*) and reputation (*Repu_wt*). RECOMMENDATION (i.e., the *R_score*) is evaluated based on value returned by the recommender (*Reco_value*) and the recommender's weight (*Recommender_wt*) according to the truster. These three values (i.e., *Evalue, Kvalue, R_score*) are normalized according to the NORMAL_POLICY. They are multiplied with their corresponding weights – *Exp_wt, Knol_wt,* and *Reco_wt*. The DYN_POLICY determines the parameter *k* to get the current value of the last available trust value. A vector from this trust history is derived and HIST_WT_POLICY specifies weights to be assigned to this vector and the former normalized vector. Composition of these two vectors results in actual trust vector with components *Exp_score, Knol_score, Reco_score* and they, in turn, return *Trust_value* between the *Truster* and the *Trustee* in a *Context* on a particular *Date*.

The relationship between different elements is summarized as follows.

1. A TRUST is always between two ENTITYs – one with a *Role* of truster and other with *Role* trustee; a truster or a trustee can be associated with zero or more TRUST.

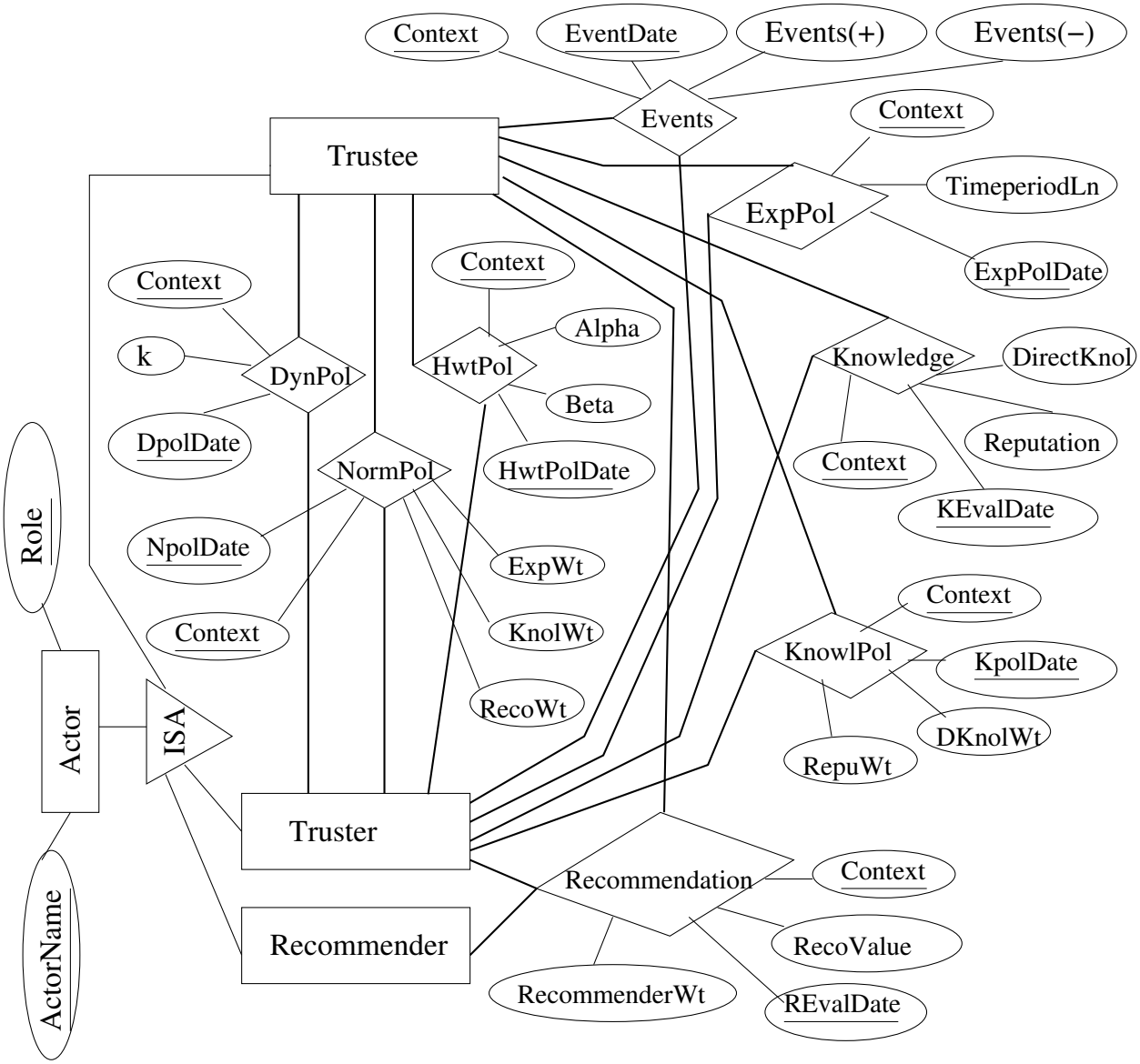


Figure 3.2: ER-diagram of the VTrust system

2. An entity can have *Role* of a truster, trustee or a recommender.

3. A TRUST is always associated with one CONTEXT; a CONTEXT can be associated with zero or more TRUST.

4. A TRUST (or, a *Trust_value*) is evaluated based on a policy; Such a policy can be associated with zero or more TRUST. A policy consists of one experience policy (EXP_POLICY), one knowledge policy (KNOW_POLICY), one normalization policy (NORMAL_POLICY), one dynamics policy (DYN_POLICY) and one history_weight policy (HIST_WT_POLICY). Each one of the parameter policy is associated with zero or more TRUST.

5. A time interval *Timeperiod* is related to zero or more EXP_POLICY.

Next we show an example of the database structure with some synthetic data. In all the tables the primary keys will be in **boldface**.

3.2.2 The VTrust database structure

For this and subsequent discussion we use a second trust relationship example. Let Alice be developing a software that has several modules with different functionality. She wants to get every module tested by an expert software engineer before she merges two modules. Assume that she assigns this testing responsibility to Bob. Thus, Alice wants to evaluate her 'trust' on Bob in the context of 'efficiency to test a software' (say, EST; acronym for the context) to decide her further course of action with Bob in the context EST. Alice sets up a trust-relationship with Bob in the context EST. She thinks of consulting Charlie, her friend and who happens to know Bob, to get his view about Bob's efficiency in this context. To store the information Alice creates the table called ENTITY as shown in Table 3.1.

Table 3.1: Initial ENTITY table

Entity_name	**Role**	**Context_name**
Alice	Truster	EST
Bob	Trustee	EST
Charlie	Recommender	EST

Let us assume that Alice starts interacting with Bob from 1st January, 2004. She decides to keep track of events occurred between her and Bob in monthly basis. Alice forms her EXP_POLICY as shown in Table 3.2.

Table 3.2: Alice's experience policy

Truster	**Trustee**	**Context**	**Epol_Date**	Timeperiod
Alice	Bob	EST	01/01/2004	1 month

Alice also sets up her knowledge policy regarding Bob. She decides to assign 70% weight on direct knowledge and 30% to indirect knowledge she gets about Bob regarding EST. Therefore, her KNOW_POLICY table looks like

Table 3.3: Alice's knowledge policy

Truster	**Trustee**	**Context**	**Kpol_Date**	Dknol_wt	Repu_wt
Alice	Bob	EST	06/30/2004	0.7	0.3

Alice can set her knowledge policy anytime before the first time she evaluates trust for Bob in EST. She can also set the normalization policy anytime prior to first evaluation of trust for Bob in EST. Let she have the NORMAL_POLICY as shown in Table 3.4.

Table 3.4: Alice's normalization policy

Truster	**Trustee**	**Context**	**Npol_Date**	Exp_wt	Knol_wt	Reco_wt
Alice	Bob	EST	10/31/2004	0.5	0.3	0.2

Now let us assume that Alice evaluates Bob's trust in EST for the first time on 31st December, 2004. On that day her EVENTS table looks like Table 3.5.

Alice next builds the EXPERIENCE, KNOWLEDGE, and RECOMMENDATION databases. The EXPERIENCE table looks like Table 3.6.

She also assigns two values for direct knowledge and reputation for Bob in EST. During the year Alice might make several visits to Bob's office to get idea about Bob's infrastructure; she checks tools and techniques used by Bob for testing. She might heard about Bob's efficiency in the job. Based on these information Alice assigns those two values according to her own judgment. The knowledge value is calculated based on the two values she provides and their corresponding weights specified in Table 3.3.

Before evaluating trust Alice consults Charlie to get his view on Bob in the context of EST. Charlie returns his judgment about Bob a recommendation value to Alice. Alice evaluates Charlie's recommendation on the basis of the trust she has on Charlie in the context of "providing a recommendation". Here we assume that a trust relationship between Alice and Charlie has already been set up prior to Bob's trust evaluation. Therefore, RECOMMENDATION table is of the form of Table 3.8.

Table 3.5: Alice's EVENTS table on 31st December, 2004

Truster	**Trustee**	**Context**	**Startdate**	**Enddate**	Events(+)	Events(-)
Alice	Bob	EST	01/01/2004	01/31/2004	3	1
Alice	Bob	EST	02/01/2004	02/29/2004	2	1
Alice	Bob	EST	03/01/2004	03/31/2004	1	3
Alice	Bob	EST	04/01/2004	04/30/2004	0	3
Alice	Bob	EST	05/01/2004	05/31/2004	2	0
Alice	Bob	EST	06/01/2004	06/30/2004	3	0
Alice	Bob	EST	07/01/2004	07/31/2004	0	2
Alice	Bob	EST	08/01/2004	08/31/2004	0	0
Alice	Bob	EST	09/01/2004	09/30/2004	0	0
Alice	Bob	EST	10/01/2004	10/31/2004	0	1
Alice	Bob	EST	11/01/2004	11/30/2004	2	0
Alice	Bob	EST	12/01/2004	12/31/2004	0	0

Table 3.6: Alice's experience value on 31st December, 2004

Truster	**Trustee**	**Context**	**Eval_Date**	Evalue
Alice	Bob	EST	12/31/2004	0.1543

Now Alice evaluates actual trust vector as well as the trust value based on these information. All these component values are normalized before calculating the trust value with the values available from Table 3.4. These calculations are internal to the system, and not of Alice's responsibility. Hence, the final TRUST table of Alice for Bob in the context EST on 31st December, 2004 is shown in Table 3.9.

Let us assume that Alice again wants to evaluate Bob after 4 months. Therefore, on 30th April, 2005 she wants to have a trust for Bob in the same context EST. We assume that after evaluating trust on 31st December, she purges all events prior to that date and start keeping log afresh. Rationale is at any later time, her decision would be influenced by the previous trust value. She does not need the whole set of events to derive current trust value. Only the events after the previous evaluation are considered to evaluate current experience. We also assume that she has not changed any policy and nothing happened between her and Bob during these 4 months. Hence the EVENTS and EXPERIENCE tables will look like Table 3.10 and Table 3.11 respectively.

Let us assume that Alice changes the values assigned to direct knowledge and reputation on the basis of her current judgment. So she adds a new entry to the KNOWLEDGE table and the table takes the form as shown in Table 3.12.

Also let us assume that the trust relationship of Alice with Charlie on the context of "providing a recommendation" changes from 0.8 to 0.7 and this time Charlie returns a lower value 0.4 for Bob

Table 3.7: Alice's knowledge value on 31st December, 2004

Truster	**Trustee**	**Context**	**Eval_Date**	Direct_knol	Reputation	Kvalue
Alice	Bob	EST	12/31/2004	0.8	0.2	0.62

Table 3.8: Alice's recommendation score on 31st December, 2004

Truster	**Trustee**	**Context**	**Eval_Date**	Recommender_name	Reco_value	Recommender_wt	R_score
Alice	Bob	EST	12/31/2004	Charlie	0.55	0.8	0.44

in the context EST. Hence we have new RECOMMENDATION table as shown in Table 3.13.

Now the trust value evaluated earlier (i.e., on 31st December, 2004) will have effect on Alice's present decision. For that Alice has to form the dynamic policy which gives the current 'level' of the previous value. Alice can form this table DYN_POLICY anytime before 30th April, 2005. Let us assume that Alice set k in DYN_POLICY as 1. This is presented in Table 3.14.

To combine the vector having current value of the parameters with the vector derived from the time-affected value of trust, Alice needs to form HIST_WT_POLICY on or before 30th April, 2005 to put relative weight on these two vectors. Let us assume that Alice put 60% weight to the vector with currently evaluated values and rest 40% to the vector derived from the time-affected value. It is shown in Table 3.15.

The final trust table on 30th April, 2005 is presented in Table 3.16.

Alice keeps on adding a new entry in the tables everytime she evaluates Bob's trust vector in EST. How long the tables should be allowed to grow depends on particular implementation and storage space available. The model does not dictate that. Instead of keeping the old records, it is quite possible to update (replace) previous record with current records. Similarly, in the policy tables old records can be kept or replaced by new records. But in TRUST table, at least the previous old record must be kept to use it later for trust dynamics.

The above calculations involved in the evaluation of components or trust value are inherent in the analysis engine which is a part of the high level system architecture for the trust management system.

Table 3.9: Alice's trust on Bob in the context EST on 31st December, 2004

Truster	**Trustee**	**Context**	**Eval_Date**	Experience	Knowledge	Reco_score	Trust_value
Alice	Bob	EST	12/31/2004	0.077	0.186	0.088	0.351

Table 3.10: Alice's EVENTS table on 30th April, 2005

Truster	**Trustee**	**Context**	**Startdate**	**Enddate**	Events(+)	Events(-)
Alice	Bob	EST	01/01/2005	01/31/2005	0	0
Alice	Bob	EST	02/01/2005	02/29/2005	0	0
Alice	Bob	EST	03/01/2005	03/31/2005	0	0
Alice	Bob	EST	04/01/2005	04/30/2005	0	0

Table 3.11: Alice's experience value on 30th April, 2005

Truster	**Trustee**	**Context**	**Eval_Date**	Evalue
Alice	Bob	EST	12/31/2004	0.1543
Alice	Bob	EST	04/30/2005	0.0

Table 3.12: Alice's knowledge value on 30th April, 2005

Truster	**Trustee**	**Context**	**Eval_Date**	Direct_knol	Reputation	Kvalue
Alice	Bob	EST	12/31/2004	0.8	0.2	0.62
Alice	Bob	EST	04/30/2005	0.9	0.1	0.66

Table 3.13: Alice's recommendation score on 30th April, 2005

Truster	**Trustee**	**Context**	**Eval_Date**	Recommender_name	Reco_value	Recommender_wt	R_score
Alice	Bob	EST	12/31/2004	Charlie	0.55	0.8	0.44
Alice	Bob	EST	04/30/2005	Charlie	0.4	0.7	0.28

Table 3.14: Alice's dynamic policy

Truster	**Trustee**	**Context**	**Dpol_Date**	k
Alice	Bob	EST	03/31/2005	1

Table 3.15: Alice's policy on assigning weights to previous trust value at current time

Truster	**Trustee**	**Context**	**Hwtpol_Date**	Alpha	Beta
Alice	Bob	EST	04/01/2005	0.6	0.4

Table 3.16: Alice's trust on Bob in the context EST on 30th April, 2005

Truster	**Trustee**	**Context**	**Eval_Date**	Experience	Knowledge	Reco_score	Trust_value
Alice	Bob	EST	12/31/2004	0.077	0.186	0.088	0.351
Alice	Bob	EST	04/30/2005	0.0064	0.4024	0.1744	0.5832

Chapter 4

TrustQL: The VTrust Query Language

Users of the VTrust trust management system need a language to interact with the system. The The language should be able to interact with the database implementation of the model. Therefore, we introduce a trust language similar to Structured Query Language (SQL). We call this language as TrustQL. The reason behind choosing this kind of language is as follows:

- We can think of the underlying data as a relational data structure. An instance of trust between truster and trustee can be saved in a row; truster, trustee, trust policy and context can be considered as individual field. Experience, knowledge and recommendation information can be considered as relations referencing corresponding relation.
- Language such as SQL processes sets of data as groups rather than as individual units. We need to consider all relevant experience, knowledge and recommendation as a whole in order to calculate the final trust level. Data as a group is more important than individual data unit.
- TrustQL uses statements that are complex and powerful individually, and that therefore stand alone.
- TrustQL differs from general purpose procedural language such as C and Java in that users specify what they want instead of how to get the result. It is up to the trust system engine to manipulate the data and present the final trust value to end users. From the user's point of view, this approach makes it easy to interact with the trust management system.

Trust query language (TrustQL) consists of Trust Definition Language (TDL) and Trust Manipulation Language (TML). TDL is used to create, alter and drop entities, policies, parameters and context. TML is used to add, modify and delete trust record as well as query the trust engine to get trust value.

Convention	*Used for*
UPPERCASE	TrustQL keywords.
italic	User-supplied parameters of TrustQL syntax.
| (vertical bar)	Separating syntax items within brackets or braces. Only one of the items can be chosen.
[ ] (brackets)	Optional syntax items. Do not type the brackets.
(braces)	Required syntax items. Do not type the braces.
[**,**...n]	Indicating that the preceding item can be repeated n number of times. The occurrences are separated by commas.
[**.**...n]	Indicating that the preceding item can be repeated n number of times. The occurrences are separated by periods.
[...n]	Indicating that the preceding item can be repeated n number of times. The occurrences are separated by blanks.
bold	Parameter names, context names, and text that must be typed exactly as shown.
<label> ::=	The name for a block of syntax. This convention is used to group and label portions of lengthy syntax or a unit of syntax that can be used in more than one place within a statement. Each location in which the block of syntax can be used is indicated with the label enclosed in angle brackets.

Table 4.1: TrustQL Syntax Convention

Trust Definition Language (TDL) consists of TrustQL keywords, Identifiers, Statements, and TrustQL convention. User-defined structures, such as entity, group and trust context name, cannot use the same identifier as keyword. The complete set of TrustQL keywords is given in section 4.1.

Definition 22 A TrustQL statement is an expression that defines a TrustQL command, such as SELECT, UPDATE, DELETE, etc. and may include clauses such as FROM, BETWEEN, AND etc. and other TrustQL keywords.

Trust Manipulation Language (TML) consists of commands like INSERT, UPDATE, DELETE, SELECT, and commands to query trust value after the trust management system has been set up using Trust Definition Language.

The conventions used in TrustQL are shown in Table 4.1.

4.1 TrustQL Keywords

Table 4.2 summarizes the TrustQL keywords. The syntax of the keywords follows in table

Table 4.2: List of TrustQL keywords

AND	ALTER	AS
ASPECT	BETWEEN	CONTEXT
CREATE	DELETE	DIRECT
DROP	DYNAMICS	ENTITY
EXPERIENCE	FROM	GROUP
HISTORY	IN	INDIRECT
INSERT	KNOWLEDGE	MEMBER
NOT	NULL	OR
POLICY	PURPOSE	RECOMMENDATION
SELECT	SET	TO
TRUST	TRSUTEE	TRSUTER
TRUSTS	VALUES	WHEN
WHO	WEIGHT	

Table 4.3: Detailed syntax of TrustQL keywords

ALTER CONTEXT { *old_context_name* } TO { *new_context_name* } [PURPOSE ({[NOT]*purpose*}[{AND | OR}{[NOT]*purpose*}])] [ASPECT ({*aspect*}[,...n])]
ALTER DYNAMICS POLICY {*trust_dynamics_name*} [TO {*new_trust_dynamics_name*}] [AS {new_integer_number}]
(*continued on next page*)

Table 4.3: (continued)

ALTER ENTITY {*old_entity_name*} TO {*new_entity_name*}
ALTER HISTORY POLICY {*old_history_policy_name*} [TO {*new_history_policy_name*}] [AS {*real_number1*, *real_number2*}]
ALTER EXPERIENCE POLICY {*experience_policy_name*} [TO {*new_experience_policy_name*}] [({ FROM *time1* TO *time2*}[,...n]) WEIGHT ({ *positive_real_number* }[,...n])]
ALTER HISTORY POLICY {*old_history_policy_name*} [TO {*new_history_policy_name*}] [AS {*real_number1*, *real_number2*}]
(*continued on next page*)

Table 4.3: (continued)

```
ALTER KNOWLEDGE POLICY
    {old_knowledge_policy_name}
    [ TO {new_knowledge_policy_name}]
    [WEIGHT
    (new_direct_knowledge_value, new_indirect_knowledge_value) ]
```

```
ALTER POLICY
    { old_policy_name }
    [ TO  {new_policy_name }]
    [WEIGHT
    {(experience_weight, knowledge_weight, recommendation_weight)}
    EXPERIENCE POLICY {experience_policy_name}
    KNOWLEDGE POLICY {knowledge_policy_name}
    RECOMMENDATION POLICY {recommendation_policy_name}
    DYNAMICS POLICY {dynamics_policy_name}
    HISTORY POLICY {history_policy_name}
    ]
```

```
ALTER RECOMMENDATION POLICY
    { recommendation_policy_name }
    [ TO {new_recommendation_policy_name}]
    [( { recommender} [,...n])
    WEIGHT
    ( { positive_real_number }[,...n] )]
```

(*continued on next page*)

Table 4.3: (continued)

```
ALTER TRUSTEE GROUP POLICY
    {group_policy_name}
    [TO {new_group_policy_name}]
    [GROUP {group_name}]
    [EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
    KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
    RECOMMENDATION WEIGHT ({member_recommentation_weight}[,n]) ]
```

```
ALTER TRUSTER GROUP POLICY
    {group_policy_name}
    [TO {new_group_policy_name}]
    [GROUP {group_name}]
    [EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
    KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
    RECOMMENDATION WEIGHT ({member_recommentation_weight}[,...n]) ]
    [POLICY {policy_name}]
```

```
CREATE CONTEXT { context_name }
    PURPOSE ({[NOT]purpose}[{ AND | OR}{[NOT]purpose}])
    ASPECT ({aspect}[,...n])
```

(*continued on next page*)

Table 4.3: (continued)

CREATE DYNAMICS POLICY {*trust_dynamics_name*} AS {*integer_number*}
CREATE ENTITY { *entity_name* }
CREATE EXPERIENCE POLICY {*experience_policy_name*} ({ FROM *time1* TO *time2*}[,...n]) WEIGHT ({ *positive_real_number* }[,...n])
CREATE GROUP {*group_name* } MEMBER ({ *entity_name* [,...n]})
CREATE KNOWLEDGE POLICY {*knowledge_policy_name*} WEIGHT (*direct_knowledge_value*, *indirect_knowledge_value*)
(*continued on next page*)

Table 4.3: (continued)

```
CREATE HISTORY POLICY
    {history_policy_name}
    AS
    {real_number1, real_number2}
```

```
CREATE POLICY
    {policy_name}
    WEIGHT
    {(experience_weight, knowledge_weight, recommendation_weight)}
    EXPERIENCE POLICY {experience_policy_name}
    KNOWLEDGE POLICY {knowledge_policy_name}
    RECOMMENDATION POLICY {recommendation_policy_name}
    DYNAMICS POLICY {dynamics_policy_name}
    HISTORY POLICY {history_policy_name}
```

```
CREATE RECOMMENDATION POLICY
    { recommendation_policy_name }
    ( { recommender} [,...n])
    WEIGHT
    ( { positive_real_number }[,...n] )
```

(*continued on next page*)

Table 4.3: (continued)

```
CREATE TRUSTEE GROUP POLICY
    {group_policy_name}
    GROUP {group_name}
    EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
    KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
    RECOMMENDATION WEIGHT ({member_recommentation_weight}[,...n])
```

```
CREATE TRUSTER GROUP POLICY
    {group_policy_name}
    GROUP {group_name}
    EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
    KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
    RECOMMENDATION WEIGHT ({member_recommentation_weight}[,...n])
    POLICY {policy_name}
```

```
DELETE TRUST
    BETWEEN {<truster>} AND {<trustee>}
    CONTEXT {context_name}
    [WHEN {some_date}]
    [WHERE <filter_expression>]

    <truster>::= {entity_name | group_name}
    <trustee>::= {entity_name | group_name}
```

(*continued on next page*)

Table 4.3: (continued)

DROP CONTEXT { *context_name* }
DROP DYANMICS POLICY {*trust_dyancmics_name*}
DROP ENTITY { *entity_name* }
DROP EXPERIENCE POLICY {*experience_policy_name*}
DROP GROUP {*group_name* }
DROP HISTORY POLICY {*history_policy_name*}
(*continued on next page*)

Table 4.3: (continued)

DROP KNOWLEDGE POLICY {*knowledge_policy_name*}
DROP POLICY { *policy_name* }
DROP RECOMMENDATION POLICY {*recommendation_policy_name*}
DROP TRUSTEE GROUP POLICY {*group_policy_name*}
(*continued on next page*)

Table 4.3: (continued)

```
INSERT TRUST
    BETWEEN {<truster>} AND {<trustee>}
    CONTEXT {context_name}
    [WHEN {some_date}]
    [EXPERIENCE VALUES {(<experience_values>)}]
    [KNOWLEDGE VALUES {(<knowledge_values>)}]
    [RECOMMENDATION VALUES {(<recommendation_values>)}]

    <truster> ::= {entity_name | group_name}
    <trustee> ::= {entity_name | group_name}
    <experience_values> ::= {time_interval, experience_value} [,...n]
    <knowledge_values> ::= {direct_knowledge_value,
                                        indirect_knowledge_value }
    <recommendation_values> ::= {<recommender>,recommendation_value}
                                                          [,...n]
    <recommender> ::= {entity_name | group_name}
```

(*continued on next page*)

Table 4.3: (continued)

```
SELECT <query_expression>
    [WHERE <filter_expression>]

    <query_expression>::= <trust_value_query_expression> |
    <trust_data_query_expression> |
    <component_query_expression>

    <trust_value_query_expression> ::= { < trust_value_attrbute_expression > }
                                                    [,...n]
    BETWEEN {<truster>} AND {<trustee>}
    CONTEXT {context_name}
    POLICY {policy_name}
    [WHEN {some_time}]

    <trust_value_attrbute_expression>::= TRUST | TRUSTER | TRUSTEE |
    TRUST VALUES | EXPERIENCE VALUES | KNOWLEDGE VALUES |
    RECOMMENDATION VALUES | CONTEXT | POLICY | POLICY VALUES

    <trust_data_query_expression>::= <trust_data_attrbute_expression>[,...n]
    BETWEEN {<truster>} AND {<trustee>}
    CONTEXT {context_name}
    [POLICY {policy_name}]
    [<time_expression>]

    <trust_data_attrbute_expression>::= TRUSTER | TRUSTEE |
    TRUST VALUES | EXPERIENCE VALUES | KNOWLEDGE VALUES |
    RECOMMENDATION VALUES | CONTEXT | POLICY | POLICY VALUES
```

(*continued on next page*)

Table 4.3: (continued)

<truster> ::= {*entity_name* | *group_name*}
<trustee> ::= {*entity_name* | *group_name*}
<time_expression>::= <time_expression1> | <time_expression2>
<time_expression1>::= WHEN {*some_time*}
<time_expression2>::= FROM {*date1*} TO {*date2*}

<component_query_expression>::= {<truster_query_expression>} |
{<trustee_query_expression>} |
{<context_query_expression>} |
{<policy_query_expression>} |
{<parameter_policy_expression>} |
{<group_policy_expression>}

<truster_query_expression>::= {<truster_attribute_expression>}[,...n]
WHO TRUSTS {<trustee>[{AND | OR}{<trustee>}][...n]}
[CONTEXT {*context_name*}]
[POLICY {*policy_name*}]
[<time_expression>]

<truster_attribute_expression>::= {TRUSTER} [CONTEXT | POLICY
|POLICY VALUES | EXPERIENCE VALUES |
KNOWLEDGE VALUES | RECOMMENDATION VALUES]

<trustee_query_expression>::= {<trustee_attribute_expression>}[,...n]
TRUSTED BY {<truster> [{AND | OR}{<truster>}][...n]}
[CONTEXT {*context_name*}]
[POLICY {*policy_name*}]
[<time_expression>]

(*continued on next page*)

Table 4.3: (continued)

```
<trustee_attribute_expression>::= {TRUSTEE} [ CONTEXT |  POLICY
| POLICY VALUES | EXPERIENCE VALUES |
KNOWLEDGE VALUES | RECOMMENDATION VALUES]

<context_query_expression>::= CONTEXT
[CONTEXT {context_name}]

<policy_query_expression>::= POLICY
[POLICY {policy_name}]

<parameter_policy_expression>::={ EXPERIENCE POLICY | KNOWLEDGE
POLICY | RECOMMENDATION POLICY | DYNAMICS POLICY | HISTORY POLICY} [,...n]
[POLICY {policy_name}]

<group_policy_expression> ::= { GROUP POLICY  }
GROUP {group_policy}
```

(*continued on next page*)

Table 4.3: (continued)

```
UPDATE TRUST
    BETWEEN {<truster>} AND {<trustee> }
    CONTEXT {context_name}
    [WHEN {some_date }]
    SET {<update_expression>}
    [WHERE <filter_expression>]

    <truster>::= {entity_name | group_name}
    <trustee>::= {entity_name | group_name}
    <update_expression>::=
    [EXPERIENCE VALUES ({ time_interval, experience_value} [,...n]) ]
    [KNOWLEDGE VALUES ({direct_knowledge_value,
                                        indirect_knowledge_value}) ]
    [RECOMMENDATION VALUES ({recommender, recommendation_value}
                                                  [,...n] )]
```

4.2 Trust Definition Language

The Trust Definition Language (TDL) is used to create, alter and drop entities, policies, parameters and context.

4.2.1 Specifying entity

Definition 23 An *entity* is any concrete or abstract thing of interest in the trust management system. An entity has a unique name.

$$\forall i, j \in \mathbb{E}, i \neq j \Leftrightarrow i.name \neq j.name$$

We use the symbol $\mathbb{E}$ to represent all entities.

Definition 24 An *active entity* is an entity who can initiate a trust evaluation process.

Definition 25 A *passive entity* is an entity that can be evaluated by other entities but can't initiate a trust evaluation process.

```
CREATE ENTITY
{ entity_name }
```

This command is used to create a new entity. *Entity_name* is always required. We assume that entity name should be unique within its namespace which can be defined when the system is implemented. We also assume that name is the only attribute of an entity although other attributes may be included during implementation phase. According to our model, an active entity or a group of active entities can act as a truster or a recommender; any entity or entity group can act as a trustee.

```
ALTER ENTITY
{old_entity_name}
TO
{new_entity_name}
```

This command is used to modify the name of an existing entity. Both *old_entity_name* and *new_entity_name* are required. The *old_entity_name* should also be unique within its namespace.

```
DROP ENTITY
{ entity_name }
```

This command is used to remove an existing entity.

Multiple entities can form a group and act as a single entity. The group can become either a truster or a trustee. Group members can be added or dropped after group formation. There must be at least one entity within a group. An entity can be members of multiple groups. A group has a name which can uniquely identify the group. Group name should be different from entity name; in other words, entity and group are in the same namespace.

```
CREATE GROUP
{group_name }
MEMBER ({ entity_name [,...n]})
```

This command is used to create a new group. *Group_name* is always required. One or more entities must be specified as group members. Multiple members are separated by comma.

```
DROP GROUP
{group_name }
```

This command is used to remove an existing group. *Group_name* is always required. If a group is removed, members in this group will become separate entities again and act independently.

4.2.2 Specifying context

The TrustQL specifications for trust context are as follows.

```
CREATE CONTEXT { context_name }
PURPOSE ({[NOT]purpose}[{ AND | OR} {[NOT]purpose}])
ASPECT ({aspect}[,...n])
```

This command is used to create a new context. *Context_name* is required and must be unique among all context names within a trust management system. At least one or more purpose must be specified. We assume that the *Purpose* and *Aspect* component have been created by the system.

```
ALTER CONTEXT { old_context_name }
TO { new_context_name }
[PURPOSE ({[NOT]purpose}[{AND | OR} {[NOT]purpose}])]
[ASPECT ({aspect}[,...n])]
```

This command is used to change the name and/or composition of an existing context. Both *old_context_name* and *new_context_name* are required. Purpose and aspect can optionally be specified if they should be modified.

```
DROP CONTEXT
{ context_name }
```

This command is used to remove an existing context. *Context_name* is required.

4.2.3 Specifying experience

```
CREATE EXPERIENCE POLICY
{experience_policy_name}
({ FROM time1 TO time2}[,...n])
WEIGHT
({ positive_real_number }[,...n])
```

This command is used to define a new experience policy and assign weight to different time intervals. *Experience_policy_name* is required and should be unique within a trust management system among all policy names. At least one interval must be specified. Multiple time intervals can not overlap. All numbers must be positive. The number of real numbers must be exactly the same as the number of intervals. The sum of all real numbers must be one.

```
ALTER EXPERIENCE POLICY
{experience_policy_name}
[TO {new_experience_policy_name}]
[ ({ FROM time1 TO time2}[,...n])
 WEIGHT
  ({ positive_real_number }[,...n])
]
```

This command is used to change the name and/or experience weight. *Experience_policy_name* is required. *New_experience_policy_name* is optional. If we want to change the experience name, we need to specify a new policy name; otherwise, leave this blank. The new value will override the previous value. The number of real numbers must be exactly the same as the number of intervals. The sum of all real numbers must be one. If we only want to change the policy name, we only need to specify the new name and ignore the rest of the command. It is invalid if both *new_experience_policy_name* and new experience weight are omitted at the same time.

```
DROP EXPERIENCE POLICY
{experience_policy_name}
```

This command is used to remove an existing experience policy. *Experience_policy_name* is required.

4.2.4 Specifying knowledge

```
CREATE KNOWLEDGE POLICY
{knowledge_policy_name}
WEIGHT
(direct_knowledge_value, indirect_knowledge_value)
```

This command is used to create a new knowledge policy and assign weight to direct knowledge and indirect knowledge. *Knowledge_policy_name* is required and should be unique within a trust

management system among all policy names. The weight for direct knowledge and indirect knowledge is assigned to *direct_knowledge_value* and *indirect_knowledge_value* respectively. Both numbers must be positive. The sum of *direct_knowledge_value* and *indirect_knowledge_value* must be one. If either one of them is not applicable, use zero as the corresponding value.

```
ALTER KNOWLEDGE POLICY
{old_knowledge_policy_name}
[ TO {new_knowledge_policy_name}]
[WEIGHT
(new_direct_knowledge_value, new_indirect_knowledge_value) ]
```

This command is used to change the name and/or weight of knowledge policy. *Old_knowledge _policy_name* is required. We can specify *new_knowledge_policy_name* if we want to change its policy name. Optionally, *new_direct_knowledge_value* and *new_indirect_knowledge_value* can be specified if we want to update their knowledge weight. The new value will override previous value. The weight for direct knowledge and indirect knowledge is assigned to *new_direct_knowledge_value* and *new_indirect_knowledge_value* respectively. Both numbers must be positive. The sum of them must remain one. If we only want to change the policy name, we can just specify *old_knowledge _policy_name* and *new_knowledge_policy_name* and ignore the rest of the command. It is invalid if both *new_knowledge_policy_name* and new knowledge weight are omitted at the same time.

```
DROP KNOWLEDGE POLICY
{knowledge_policy_name}
```

This command is used to remove an existing knowledge policy. *knowledge_policy_name* is required.

4.2.5 Specifying recommendation

```
CREATE RECOMMENDATION POLICY
{ recommendation_policy_name }
( { recommender} [,...n])
WEIGHT
( { positive_real_number }[,...n] )
```

This command is used to define a new recommendation policy and assign weight to various recommenders. *Recommendation_policy_name* is required and must be unique within a trust man-

agement system among all policy names. At least one recommender must be specified. All numbers must be positive. The number of real number must be exactly the same as the number of recommenders. The sum of all real numbers must be one.

```
ALTER RECOMMENDATION POLICY
{ recommendation_policy_name }
[ TO {new_recommendation_policy_name}]
[( { recommender} [,...n])
WEIGHT
( { positive_real_number }[,...n] )]
```

This command is used to change recommendation policy name or the recommender weight. *Recommendation_policy_name* is required. *New_recommendation_policy_name* is needed only if we want to change the policy name. The new value will override the previous value. The number of real number must be exactly the same as the number of recommenders. The sum of all real numbers must remain one. If we only want to change the policy name, we can only specify the new name and ignore the rest of the command. It is invalid if both *new_recommendation_policy_name* and new recommendation weight value are omitted at the same time.

```
DROP RECOMMENDATION POLICY
{recommendation_policy_name}
```

This command is used to remove an existing recommendation policy. The *recommendation_policy_name* is required.

4.2.6 Specifying trust dynamics

```
CREATE DYNAMICS POLICY
{trust_dynamics_name}
AS
{integer_number}
```

This command is used to define a new trust dynamic policy and assign an integer value to *k*. *Trust_dynamics_name* is required and should be unique within a trust management system among all policy names. *Integer_number* are required.

```
ALTER DYNAMICS POLICY
{trust_dynamics_name}
[TO {new_trust_dynamics_name}]
[AS
{new_integer_number} ]
```

This command is used to modify the name and/or value of an existing trust dynamics policy. *Trust_dynamics_name* is required. *New_trust_dynamics_name* is optional. If we want to change the name of the policy, specify the new name; otherwise, omit the new name. The *new_integer_number* is also optional. Specify the number only if we want to change the trust dynamics value. It is invalid if both *new_trust_dynamics_name* and *new_integer_number* are omitted at the same time.

```
DROP DYANMICS POLICY
{trust_dyancmics_name}
```

This command is used to remove an existing trust dynamics policy. *Trust_dynamics_name* is required.

```
CREATE HISTORY POLICY
{history_policy_name}
AS
{real_number1, real_number2}
```

This command is used to create a new history policy. *history_name* is required and should be unique within a trust management system among all policy names. *Real_number1* and *real_number2* are also required. The value of α and β is assigned to *real_number1* and *real_number2* respectively. Both *real_number1* and *real_number2* should be positive and the sum should be one.

```
ALTER HISTORY POLICY
{old_history_policy_name}
[TO {new_history_policy_name}]
[AS
{real_number1, real_number2} ]
```

This command is used to modify the name and/or value of an existing history policy. *Old_history_policy_name* is required. *New_history_policy_name* is optional. If we want to change the name of

the policy, specify the new name; otherwise, omit the new name. *real_number1* and *real_number2* are also optional. Specify the number only if we want to change the α and β value. It is invalid if both *new_history_policy_name* and the two numbers are omitted at the same time.

```
DROP HISTORY POLICY
{history_policy_name}
```

4.2.7 Specifying trust evaluation policy

```
CREATE POLICY
{policy_name}
WEIGHT
{(experience_weight, knowledge_weight, recommendation_weight)}
EXPERIENCE POLICY {experience_policy_name}
KNOWLEDGE POLICY {knowledge_policy_name}
RECOMMENDATION POLICY {recommendation_policy_name}
DYNAMICS POLICY {dynamics_policy_name}
HISTORY POLICY {history_policy_name}
```

This command is used to create a new trust policy which specifies the weight of three parameters, time dynamics policy and history policy. *Policy_name* is required and should be unique among all policies. *Experience_weight*, *knowledge_weight* and *recommendation_weight* are required and are assigned as experience, knowledge and recommendation weight respectively. The order is fixed. These three numbers must be positive; the sum of them must be one. When defining a policy, we must associate an existing experience policy, knowledge policy, recommendation policy, dynamics policy and history policy with it.

```
ALTER POLICY
{ old_policy_name }
[ TO  {new_policy_name }]
[WEIGHT
{(experience_weight, knowledge_weight, recommendation_weight)}
EXPERIENCE POLICY {experience_policy_name}
KNOWLEDGE POLICY {knowledge_policy_name}
RECOMMENDATION POLICY {recommendation_policy_name}
DYNAMICS POLICY {dynamics_policy_name}
```

```
HISTORY POLICY {history_policy_name}
]
```

This command is used to modify the name of an existing trust policy and/or the weight of parameters. *Old_policy_name* is required. *New_policy_name* is optional. *New_policy_name* is not needed if the policy name is not changed. *Experience_weight*, *knowledge_weight* and *recommendation_weight* are assigned as experience, knowledge and recommendation weight respectively. The order is fixed. All three numbers are required and must be positive. The sum of all three numbers must be one. If we only want to change the policy name, we just need to specify *old_policy_name* and *new_policy_name* and ignore the rest of the command. Parameter policy can also be modified. It is invalid if *new_policy_name* and parameter policy are omitted at the same time.

When modifying an existing policy, all components in this policy must be explicitly listed even if its weight is not changed. If a component in an existing policy is not listed when updating the policy, its weight will be dropped.

```
DROP POLICY
{ policy_name }
```

This command is used to remove an existing policy. *Policy_name* is required.

4.2.8 Specifying trustee group policy

```
CREATE TRUSTEE GROUP POLICY
{group_policy_name}
GROUP {group_name}
EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
RECOMMENDATION WEIGHT ({member_recommentation_weight}[,...n])
```

This command is used to create a new trustee group policy. *Group_policy_name* is required. *Group_name* is also required which refers to an existing group. *Member_experience_weight* is the weight for individual group members regarding experience. The sum of all *member_experience_weight* must be one. The number of *member_experience_weight* must be the same as the number of group members. The same is true for knowledge weight and recommendation weight.

```
ALTER TRUSTEE GROUP POLICY
{group_policy_name}
[TO {new_group_policy_name}]
[GROUP {group_name}]
[EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
RECOMMENDATION WEIGHT ({member_recommentation_weight}[,n]) ]
```

This command is used to modify an existing trustee group policy. Group policy name, corresponding group and member weight can be changed. *Group_policy_name* is required. *New_group _policy_name*, *group_name* and member weight are optional.

```
DROP TRUSTEE GROUP POLICY
{group_policy_name}
```

This command is used to remove an existing trustee group policy. *Group_policy_name* is required.

4.2.9 Specifying truster group policy

```
CREATE TRUSTER GROUP POLICY
{group_policy_name}
GROUP {group_name}
EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
RECOMMENDATION WEIGHT ({member_recommentation_weight}[,...n])
POLICY {policy_name}
```

This command is used to create a new truster group policy. *Group_policy_name* is required. *Group_name* is also required which refers to an existing group. *Member_experience_weight* is the weight for individual group member regarding experience. The sum of all *member_experience_weight* must be one. The number of *member_experience_weight* must be the same as the number of group members. The same is true for knowledge weight and recommendation weight. *Policy_name* is required and used to specify the common policy used by all group members.

```
ALTER TRUSTER GROUP POLICY
{group_policy_name}
```

```
[TO {new_group_policy_name}]
[GROUP {group_name}]
[EXPERIENCE WEIGHT ({member_experience_weight}[,...n])
KNOWLEDGE WEIGHT ({member_knowledge_weight}[,...n])
RECOMMENDATION WEIGHT ({member_recommentation_weight}[,...n]) ]
[POLICY {policy_name}]
```

This command is used to modify an existing truster group policy. *Group_policy_name*, corresponding group, member weight and common policy can be changed. *Group_policy_name* is required. *New_group_policy_name*, *group_name*, member weight and *policy_name* are optional.

```
DROP TRUSTER GROUP POLICY
{group_policy_name}
```

This command is used to remove an existing truster group policy. *Group_policy_name* is required.

4.2.10 Specifying group reconfiguration

After the group is formed, some members may leave, or some new members may join the group. This contraction (or, expansion) of the group can happen in steps or, in one instance. That is, old members can leave one by one or, together. Similarly, new members can join in subsequent time instances, or as a whole group. Reconfiguration of a group is specified according to TDL as follows:

```
ALTER GROUP
{old_group_name }
[TO {new_group_name }]
[ADD MEMBER ({entity_name}[,...n]) |
 DROP MEMBER ({entity_name}[,...n]) ]
```

This command is used to modify the name of an existing group or to change group members. *Old_group_name* is required. The new group name should be unique within entity and group namespace. Optionally, group membership can be changed. We can either add new group members or drop existing members. Multiple entity names are separated by comma.

4.3 Trust Manipulation Language

Trust Manipulation Language (TML) is used to insert, update, delete and query trust value after the trust management system has been set up using Trust Definition Language (TDL).

4.3.1 INSERT trust value

```
INSERT TRUST
BETWEEN {<truster>} AND {<trustee>}
CONTEXT {context_name}
[WHEN {some_date}]
[EXPERIENCE VALUES {(<experience_values>)}]
[KNOWLEDGE VALUES {(<knowledge_values>)}]
[RECOMMENDATION VALUES {(<recommendation_values>)}]

<truster> ::= {entity_name | group_name}
<trustee> ::= {entity_name | group_name}
<experience_values> ::= {time_interval, experience_value} [,...n]
<knowledge_values> ::= {direct_knowledge_value,
                                    indirect_knowledge_value }
<recommendation_values> ::= {<recommender>, recommendation_value}
                                                    [,...n]
<recommender> ::= {entity_name | group_name}
```

This command is used to insert trust data into the trust management system. Both truster and trustee are required. *Context_name* is also required which specifies the context under which the trust data was obtained. Date is optional. When *some_date* is omitted, the current date is used as the default date. If *some_date* is specified, then the data is considered to be obtained at that time. Only past date can be specified, future date is not allowed as future trust data can't be predicted. *Experience_value* can be NULL if the actual value is not available.

4.3.2 UPDATE trust value

```
UPDATE TRUST
BETWEEN {<truster>} AND {<trustee> }
CONTEXT {context_name}
```

```
[WHEN {some_date }]
SET {<update_expression>}
[WHERE <filter_expression>]

<truster>::= {entity_name | group_name}
<trustee>::= {entity_name | group_name}
<update_expression>::=
[EXPERIENCE VALUES ({ time_interval, experience_value} [,...n]) ]
[KNOWLEDGE VALUES ({direct_knowledge_value,
                                    indirect_knowledge_value}) ]
[RECOMMENDATION VALUES ({recommender, recommendation_value}[,...n] )]
```

This command is used to modify existing trust data. Both truster and trustee are required. *Context_name* is also required which specifies the context under which the trust data was obtained. Date is optional. When *some_date* is omitted, all trust data between truster and trustee under the context will be updated; if *some_date* is specified, then the trust data at the specified time is updated. The format of experience, knowledge and recommendation value is the same as insert command. The Where clause is optional; it is used to restrict the trust data to be updated. Filter_expression specifies the condition to be met for the trust data to be updated. There is no limit to the number of predicates that can be included in the condition.

4.3.3 DELETE trust value

```
DELETE TRUST
BETWEEN {<truster>} AND {<trustee>}
CONTEXT {context_name}
[WHEN {some_date}]
[WHERE <filter_expression>]

<truster>::= {entity_name | group_name}
<trustee>::= {entity_name | group_name}
```

This command is used to delete existing trust data. Both truster and trustee are required. *Context_name* is also required to specify the context between truster and trustee. When clause is optional. If when clause is omitted, then all trust data between the truster and trustee under the specified context will be deleted; otherwise, only the trust data happened at the specified time

is deleted. Where clause is optional. Filter_expression specifies the condition to be met for the trust data to be deleted. There is no limit to the number of predicates that can be included in the condition.

4.3.4 SELECT trust value

Applications interacting with trust management system sends query to the trust management system. A trust query is essentially a string. The trust evaluation engine will evaluate the query after the string is parsed and validated. All aspect of the trust management system can be queried such as trust value, entity, context, policy, etc.

```
SELECT <query_expression>
[WHERE <filter_expression>]

<query_expression>::= <trust_value_query_expression> |
<trust_data_query_expression> |
<component_query_expression>

<trust_value_query_expression> ::= { < trust_value_attrbute_expression > }
                                                                  [,...n]
BETWEEN {<truster>} AND {<trustee>}
CONTEXT {context_name}
POLICY {policy_name}
[WHEN {some_time}]

<trust_value_attrbute_expression>::= TRUST | TRUSTER | TRUSTEE |
TRUST VALUES | EXPERIENCE VALUES | KNOWLEDGE VALUES |
RECOMMENDATION VALUES | CONTEXT | POLICY | POLICY VALUES

<trust_data_query_expression>::= <trust_data_attrbute_expression>[,...n]
BETWEEN {<truster>} AND {<trustee>}
CONTEXT {context_name}
[POLICY {policy_name}]
[<time_expression>]
```

<trust_data_attrbute_expression>::= TRUSTER | TRUSTEE |
TRUST VALUES | EXPERIENCE VALUES | KNOWLEDGE VALUES |
RECOMMENDATION VALUES | CONTEXT | POLICY | POLICY VALUES

<truster> ::= {*entity_name* | *group_name*}
<trustee> ::= {*entity_name* | *group_name*}
<time_expression>::= <time_expression1> | <time_expression2>
<time_expression1>::= WHEN {*some_time*}
<time_expression2>::= FROM {*date1*} TO {*date2*}

<component_query_expression>::= {<truster_query_expression>} |
{<trustee_query_expression>} |
{<context_query_expression>} |
{<policy_query_expression>} |
{<parameter_policy_expression>} |
{<group_policy_expression>}

<truster_query_expression>::= {<truster_attribute_expression>}[,...n]
WHO TRUSTS {<trustee>[{AND | OR}{<trustee>}][...n]}
[CONTEXT {*context_name*}]
[POLICY {*policy_name*}]
[<time_expression>]

<truster_attribute_expression>::= {TRUSTER} [CONTEXT | POLICY
|POLICY VALUES | EXPERIENCE VALUES |
KNOWLEDGE VALUES | RECOMMENDATION VALUES]

<trustee_query_expression>::= {<trustee_attribute_expression>}[,...n]
TRUSTED BY {<truster> [{AND | OR}{<truster>}][...n]}
[CONTEXT {*context_name*}]
[POLICY {*policy_name*}]
[<time_expression>]

<trustee_attribute_expression>::= {TRUSTEE} [CONTEXT | POLICY

```
| POLICY VALUES | EXPERIENCE VALUES |
KNOWLEDGE VALUES | RECOMMENDATION VALUES]

<context_query_expression>::= CONTEXT
[CONTEXT {context_name}]

<policy_query_expression>::= POLICY
[POLICY {policy_name}]

<parameter_policy_expression>::={ EXPERIENCE POLICY | KNOWLEDGE
POLICY | RECOMMENDATION POLICY | DYNAMICS POLICY | HISTORY POLICY} [,...n]
[POLICY {policy_name}]

<group_policy_expression> ::= { GROUP POLICY  }
GROUP {group_policy}
```

This command is used to query final trust value, trust data, context, policy, truster, trustee, context and parameter policy information. The command can take many different forms. There are several patterns that we should follow when formulating the command.

- When querying final trust value, we must specify truster, trustee, context and policy. If when clause is omitted, the current trust is evaluated, otherwise, trust value is evaluated at the specified point of time. This time can't be a future date.
- Trust fact data can be returned without specifying trust policy. Two kinds of formats can be used. The first one is a specified point of time; the second one is a time interval from two dates.
- Context, policy and parameter policy can be returned directly.
- Trust group can act as a single truster or a trustee. Whenever a group is involved in a trust relationship, group policy is used.
- We have defined the return information without specifying the return format. This could be an implementation decision.
- We define the basic trust query syntax here; however, several queries can be combined during implementation time. For example, we can query CONTEXT and POLICY directly.

- The above syntax is used by the trust engine. Another layer will be built if the engine interacts with other systems. In this case, the trust engine acts as a server and other systems act as clients. The syntax that the clients used will be relatively easy to use and may contain graphical user interface. Nevertheless, the query string that other applications generated will have to be transformed into such syntax that the trust engine can understand.

Chapter 5

Model Application

The vector trust model is geared towards enhancing security in systems. We investigated how to apply this model for such purposes. The problem domain of interest is that of access control in open and distributed environments. The NAS system of the Federal Aviation Administration is an example of such a system.

Proper access control to resources is one of the major security concerns for any organization. Different models of access control have been proposed over the years, for example, discretionary and mandatory access control models, Clark-Wilson model, Task based models and Role Based Access Control model. Among these, role based access control (RBAC) [4] is gradually emerging as the standard for access control. The main advantage of RBAC over other access control models is the ease of security administration. In the RBAC model access permissions are not assigned directly to the users but to abstractions known as "roles". Roles correspond to different job descriptions within an organization. Users are assigned to different roles and, thus, indirectly receive the relevant permissions. Thus, with RBAC, security is managed at a level corresponding to an organization's human resource structure.

Notwithstanding the success of the RBAC model, researchers have often found the model to be inadequate for open and decentralized multi-centric systems where the user population is dynamic and the identity of all users are not known in advance. Examples of such systems are service providers operating over open systems like the Internet. It is almost impossible to know beforehand all the users that will request services in these systems. Assigning appropriate roles to these users thus becomes an irrational and ad-hoc exercise. To overcome the shortcomings of RBAC for such systems, researchers have proposed credential-based access control models [2, 1, 7]. Credentials implement a notion of binary trust. Here the user has to produce a pre-determined set of credentials (for example, credit card numbers or proof of membership to certain groups etc.) to

gain specific access privileges. The credential provides information about the rights, qualifications, responsibilities and other characteristics attributable to its bearer by one or more trusted authorities. In addition, it provides trust information about the authorities themselves. Researchers have also integrated credential based access control with role-based access control to facilitate security administration [9, 3, 8, 10].

Although credential based models solve the problem of access control in open systems to a great extent, it still has a number of shortcomings. A credential, strictly speaking, does not bind a user to its purported behavior or actions. It does not guarantee that its bearer really satisfies the claims in the credential. It does not convey any information about the behavior of the bearer between the time the credential was issued and its use. A credential does not reveal whether it was obtained via devious means. In real life some or all such information may play crucial parts in access control decisions. Additionally, credential based access control does not keep track of the user's behavior history. Access permission is given on the basis of the credential presented for a particular session. Either the user's credentials are accepted and required privileges are allowed, or the credentials are rejected and the user does not get the access rights. Thus, good behavior by the user cannot be rewarded with enhanced privileges nor bad behavior be punished.

The above observations motivate us to revisit the problem of access control in decentralized and multi-centric open systems. We believe credential based access control is a step in the right direction. However, we would like to enhance the binary trust paradigm in these models with the much richer multi-level trust model. In our trust model, trust levels in the users can be determined not only by using the credentials presented by the user but also from the results of past interactions with the user, from recommendations about the user and/or knowledge about other characteristics of the user. A user is mapped to different trust levels based on these information. Trust levels (and not users, unlike in conventional RBAC) are then mapped to roles of RBAC. Thus our access control model is an enhanced RBAC (TrustBAC). Changes in the trust level of user changes the roles that the user has in the system and thus the user's privileges. The system can define as many trust levels as it wants and can assign each level to a specific set of resources tied with a specific set of access privileges. The system just needs to monitor the trust level of the user and the regulation of access is automatically achieved.

5.1 TrustBAC model

The TrustBAC model is defined in terms of a set of elements and relations among those elements. The elements are of the following types: *user, user_properties, session_instance, session_type, ses-*

sion, session_history, trust_level, role, object, action, permissions and *constraint*. The corresponding sets are USERS, USER_PROPERTIES, SESSION_INSTANCES, SESSION_TYPES, SESSIONS, SESSION_HISTORY, TRUST_LEVELS, ROLES, OBJECTS, ACTIONS, PERMISSIONS and CONSTRAINTS. The TrustBAC model is illustrated in figure 5.1(we use one-directional arrows to represent one-to-many relationship, two directional arrows to denote many-to-many relationships and plain lines to denote one-to-one relationships). We define the different elements as follows.

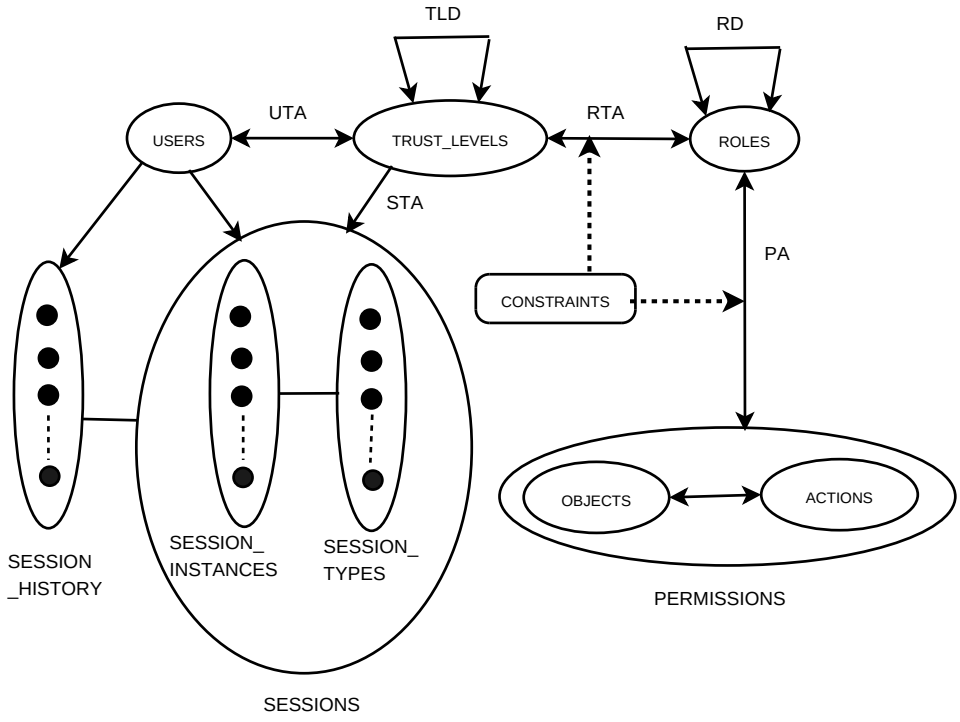


Figure 5.1: TrustBAC model

user A user $\in$ USERS is defined as a human being. The notion of user can be extended to include systems, or intelligent agents, but for simplicity we choose to limit a *user* to a human entity.

user_properties Each user u has certain set of properties $\mathscr{P}_u$, called *user_properties*. The set USER_PROPERTIES = $\bigcup_{u \in USERS} \mathscr{P}_u$. A user can manifest any subset P of $\mathscr{P}_u$ (i.e., $P \in 2^{\mathscr{P}_u}$) at a particular session.

session_instance A session_instance $\in$ SESSION_INSTANCES is a 'login' instance of an user. A user can instantiate multiple login thereby initiating multiple session_instances at the same time. A session_instance is uniquely identified by a system generated id.

session-type A session_instance is identified with a type which is determined by the set of properties manifested in that session_instance by the user invoking that session_instance. For a session_instance s invoked by a user u with P ($P \subseteq \mathscr{P}_u$) properties, has the session_type P. Formally, the set SESSION_TYPES = $2^{USER\_PROPERTIES}$.

session A session $\in$ SESSIONS is identified by a session_instance with a session_type. A session with session_instance s of type P is denoted by the symbol s^P. Formally, $SESSIONS = SESSION\_INSTANCES \times SESSION\_TYPES$.

session_history A session_history $\in$ SESSION_HISTORY is a set of information regarding the user's behavior and trust level in a previous use of a session of that type.

trust_level A trust_level is a set of real number between -1 and +1. A user, at some instant of time with a particular session has a trust_level. The set TRUST_LEVELS is the set of possible subsets of [-1, 1]. That is, TRUST_ LEVELS = $\{S \mid S \subseteq [-1,1]$. Thus TRUST_LEVELS becomes an infinite set where each member S can be either discrete or continuous.

role The concept of role is same as in the RBAC model. A role $\in$ ROLES is a job function with some associated semantics regarding the responsibilities conferred to a user assigned to the role.

object An object $\in$ OBJECTS is a data resource as well as a system resource. It can be thought of as a *container* that contains information.

action An action $\in$ ACTIONS is an executable image of a program. 'read', 'write', 'execute' are examples of a typical action.

permission A permission $\in$ PERMISSIONS is an authorization to perform certain task within the system. It is defined as a subset of OBJECTS $\times$ ACTIONS i.e., $PERMISSIONS = 2^{(OBJECTS \times ACTIONS)}$. Therefore, a permission = $\{(o,a) \mid o \in OBJECTS, a \in ACTIONS)$. Permissions are assigned to a role. The type of a permission depends on the nature of the system. The model does not dictate anything about the type.

constraint We borrow the concept of constraint from RBAC model. Therefore, a constraint $\in$ CONSTRAINTS is defined as a predicate which applied to a relation between two TrustBAC elements returns a value of "acceptable" or "not-acceptable". Constraints can be viewed as conditions imposed on the relationships and assignments.

Association between any two of the above elements are specified by mathematical relations. TrustBAC has the following relations.

1. *sua* : USERS $\times$ SESSION_INSTANCES $\times$ SESSION_TYPES $\rightarrow$ SESSIONS defines the *user-session* assignment relation. $sua(u,s,P) = s^P$ for u $\in$ USERS, s $\in$ SESSION_INSTANCES, P $\in$ SESSION_TYPES, and $s^P \in$ SESSIONS shows that a single session s^P of type P is associated with a single user u with certain properties P. A user can invoke multiple sessions of different types simultaneously.

2. UTA $\subseteq$ USERS $\times$ TRUST_LEVELS defines the *user-trust_level* assignment relation. It is a many-to-many relation where a user can have multiple trust levels. Since a user can invoke many sessions at a time, she can have different trust levels, one for each invoked session. A single trust_level can be assigned to many users. The restriction on a member $(u,L) \in$ UTA is L must be a singleton member of TRUST_LEVELS i.e., $L = \{l\},\ l \in [-1,1]$.

3. STA $\subseteq$ SESSIONS $\times$ TRUST_LEVELS defines the *session-trust_level* assignment. It is a one-to-many relation where a session can have only one trust value. That is, the trust_level L corresponding to that session is a singleton member of TRUST_LEVELS. But many sessions can have the same trust_level.

4. RTA $\subseteq$ ROLES $\times$ TRUST_LEVELS defines the *role-trust_level* assignment relation. It is also a many-to-many relationship where a trust_level can be associated with many roles and same role can be performed with different trust_levels.

5. The function *ush*: USERS $\times$ SESSION_TYPES $\rightarrow$ SESSION_HISTORY defines a three-way relation between a user, a session_type and the trust history of the user in an earlier use of a session of that type. $ush(u,P) = {}_uh^P$, where u $\in$ USERS and P $\in$ SESSION_TYPES. A session_history ${}_uh^P$ is associated with a single user u and any session s^P of type P. A user can have many session_histories as a user can invoke many sessions of different types.

6. PA $\subseteq$ PERMISSIONS $\times$ ROLES is a many-to-many permission to role assignment relation. An element in PA is of type (p,r) where $p \in$ PERMISSIONS and $r \in$ ROLES.

7. The function $Assigned\_Roles : TRUST\_LEVELS \rightarrow 2^{ROLES}$ specifies the mapping of a trust_level $L(\subseteq [-1,1])$ onto a set of roles. Formally, $Assigned\_Roles(L) = \{r \in ROLES \mid (r,L) \in RTA\}$. It implies, for any $l \in L, Assigned\_Roles(\{l\}) = Assigned\_Roles(L)$.

8. The function $Assigned\_Permission : ROLES \rightarrow 2^{PERMISSIONS}$ specifies the mapping of a role r onto a set of permissions. Formally, $Assigned\_Permission(r) = \{p \in PERMISSIONS \mid (p,r) \in PA\}$. This function is same as *assigned_permissions* function of RBAC model.

The constraints are applied on the above assignment functions depending on the access control policies of the system. Constraints on *Assigned_Roles* are similar to the constraints on user-role assignment in RBAC model. It specifies which roles are 'permitted' to be assigned to a certain trust_level. Constraints on *Assigned_Permissions* determines the assignment of permissions to a specific role. RBAC model suggests different constraints like *mutually exclusive role, prerequisite roles, cardinality constraints, static separation of duty, dynamic separation of duty* etc. But we prefer not to specify any particular constraint on these functions. Rather we leave it as general to give finer control in defining access control policies depending on the requirements of a system.

We also introduce a concept of **role dominance** among roles in our model. *Role dominance* is similar to the concept of *role hierarchies* in RBAC model. A role dominance relation, denoted by RD, defines a dominance relation between two roles. The dominance is described in terms of permissions. We define role dominance as,

Definition 26 Role dominance $RD \subseteq ROLES \times ROLES$ is a partial order on ROLES where the partial order is called a *Dominance* relation, denoted by $\preceq$. For any $(r_1, r_2) \in RD$, we say r_2 'dominates' r_1 only if all permissions assigned to r_1 are also permissions of r_2. Formally, $(r_1, r_2) \in RD \Rightarrow r_1 \preceq r_2$ and $r_1 \preceq r_2 \Rightarrow Assigned\_Permissions(r_1) \subseteq Assigned\_Permission(r_2)$.

The above definition implies that any user u having a role r_2 can have all the privileges of a user with role r_1.

The relation RD induces a similar relation called **trust dominance** among trust_levels in our model. Whenever there is a role dominance between two roles, there is a trust_level dominance between the corresponding trust_levels. Trust_level dominance, denoted by TLD is defined as follows:

Definition 27 Trust_level dominance, $TLD \subseteq TRUST\_LEVELS \times TRUST\_LEVELS$ is a partial order relation on TRUST_LEVELS and is denoted by $\leq'$. For any $(L_1, L_2) \in TLD$, we say L_2 'dominates' L_1 only if $L_1 \subseteq L_2$. If L_2 is a singleton set $\{l_2\}$, then dominance is defined as, $sup\{L_1\} \leq l_2$ that is, l_2 is greater than or equal to the maximum element of L_1. If both $L_1 = \{l_1\}$ and $L_2 = \{l_2\}$ are singletons then $L_1 \leq' L_2 \Rightarrow l_1 \leq l_2$ (the $\leq$ is the usual 'less equalto' relation of number theory.)

The relation TLD is induced by RD. That is, for any $(r_1, r_2) \in RD, \exists (L_1, L_2) \in TLD$ such that $r_1 \in Assigned\_Roles(L_1)$ and $r_2 \in Assigned\_Roles(L_2)$. That is, the trust degree of a user with role r_2 is greater than that of a user with role r_1.

5.2 Access Control Using TrustBAC

Basic purpose of an access control mechanism is to protect system resources by restricting the user's activities on them. A user's authorization to perform certain tasks on specific resources is specified by the access control policy of the system. When using TrustBAC for access control, a *user* invokes a *session_instance* of a particular type at an instant of time. During this session the user has a *trust_level* which allows her to use the *roles* associated with that trust_level. That is, a user can be a member of a role. Also a single role can be exercised by many users. For each of these roles, the user has a set of *permissions*. Therefore, the user is restricted to perform a set operations on a particular set of resources as specified by the set of permissions obtained as a member of those roles.

A first time user u registers with the system and logs in which instantiates a session_instance s of the user. Depending on the set of disclosed properties P, the system invokes the function *sua* with arguments u, s, and P to start a session s^P. The system initiates a trust relationship $(SYS \xrightarrow{P} u)_t^N$ with the user in that session. The underlying context of this trust relationship is identified by the session_type P. This relationship does not change, but gets updated for any other session of same type P invoked by the same user u. If the user invokes another session_instance of type P$'$ at time t, then the system creates another trust relationship $(SYS \xrightarrow{P'} u)_t^N$. The value of the trust relationship $(SYS \xrightarrow{P} u)_t^N$ is evaluated for the session s^P. Let $\mathbf{v}(SYS \xrightarrow{P} u)_t^N = l,\ l \in [-1,1]$. The system invokes the function *Assigned_Roles* to determine the roles that the user u can execute. Let $Assigned\_Roles(\{l\}) = \{r_1, r_2, \ldots, r_n\}$. u can choose to execute more than one of these n roles. With each r_i, u has a set of p_js where $\forall j, (p_j, r_i) \in PA$. Therefore, in a session s^P, the user u has the set of permissions given by, $\bigcup_i Assigned\_Permissions(r_i) = \bigcup_{1 \le i \le n} \{p_{ji} \mid (p_j, r_i) \in PA\}$. Hence, the user u is restricted to perform actions A on a set of objects O where $\exists$ a $p_{ji} \in \bigcup_{1 \le i \le n} \{p_{ji} \mid (p_j, r_i) \in PA\}$, such that, for any $(o,a) \in O \times A,\ (o,a) \in p_{ji}$. The user executes these actions on the allowed objects and each activity during that session s^P is stored as the session_history ${}_uh^P$ for that session. Whenever the trust_level is re-evaluated (within s^P or, at the start of next instance s$'$ of a session of type P), the *events* in ${}_uh^P$ are evaluated. The evaluated trust_level, say l' overwrites l in ${}_uh^P$. The subsequent events also overwrite the previous event log.

We assume that for a registered user u in a session s^P, the trust relationship $(SYS \xrightarrow{P} u)$ is managed by a diligent system-administrator who is outside the scope of this access control framework. We denote this system-admin by the symbol SYS. We also assume that the system has two pre-defined policies – an access control policy $\mathcal{A}_{sys}$ and a trust evaluation policy $\mathcal{T}_{sys}$ which are not independent. $\mathcal{A}_{sys}$ defines the functions *Assigned_Roles* and *Assigned_Permissions* together with the 'constraints' on them. The components are evaluated as

Computing *knowledge* The user u initiates the session s^P by disclosing a set of properties P, which includes information (e.g., name, address, affiliation, etc.) as well as some credentials. Credentials are in the form of typical digital certificates. The system assign a value within $[-1,1]$ as weights to the information and the credentials. The assignment is done as specified by $\mathcal{T}_{sys}$ and ${}_{SYS}K_u^P$ is computed according to the equation (2.5). The next instance of a session of type P, the values assign to members of P may change due to change in values in P. For example, the user disclose the same type of certificate, but with a different certifying authority.

Computing *experience* As mentioned in section 2.2.2, experience is computed from the *events* occurred during some intervals. Our model does not dictate about the length of an interval. It depends on implementation – the system may choose to identify a whole session as an interval. Independent of the length of an interval, any *action* performed by the user is identified as an 'event'. This record is kept in session_history ${}_uh^P$ till the next instant of trust evaluation. Formally, let l be the trust_level of u in a session s^P. Let $Assigned\_Roles(l) = \{r_1, r_2, \ldots, r_n\}$ of which u activate r_1, r_2, r_3. These are the *active roles* of u in session s^P. The events are the set of actions $\mathbb{A}$ where for any $a \in \mathbb{A}$, $\exists\ p \in \bigcup_{1 \le i \le 3} Assigned\_Permissions(r_i)$. The weight to the result of a particular action is assigned according to $\mathcal{T}_{sys}$ and the experience ${}_{SYS}E_u^P$ is computed as specified by equation (2.4).

Computing *recommendation* The system may take *role-specific* and *role-independent* input from other users about u in a session. These information constitute u's recommendation and the component ${}_\Psi R_u^P$ is calculated using equation in (2.6). Ψ is the set of other users who provide recommendation for u to SYS. However, we choose not to specify how these information are collected.

After computing the components, the system calculates the normalized trust by combining $(SYS \xrightarrow{P} u)_t$ and the normalization policy of $\mathcal{T}_{sys}$. Then the previous trust_level is fetched from ${}_uh^P$ and final $(SYS \xrightarrow{P} u)_t^N$ is calculated using the equation (2.7). The corresponding value $\mathbf{v}(SYS \xrightarrow{P} u)_t^N$ is calculated as specified in the section 2.2.6. This value denotes the current trust_level of u in a session of type P and gets stored in corresponding session_history ${}_uh^P$.

Chapter 6

Conclusions and Future Work

The concept of trust is widely used in secure information systems. For example, “trusted computing base” refers to the hardware and software that make up the security of a system; a “trusted system” is one that is believed to be secure against relevant attacks, and so on. However, until recently, there were no accepted formalism or techniques for the specification of trust and for reasoning about it. Secure systems had been built under the premise that concepts like “trusted” or “trustworthiness” were well understood, unfortunately without even agreeing on what “trust” means, how to measure it, how to compare two trust values and how to compose the same. There had been a lack of a comprehensive mathematical framework for quantifying the amount of trust that can be placed on complex systems. There are two models of trust widely used Ð the binary models and the non-binary models. The binary models assign a value to trust of either 0 (no trust) or 1 (complete trust). The non-binary models assign quantitative values in the range 0 to 1 or qualitative values like high, medium, or low. The values of these models tend to be subjective estimates. The existing models have no accepted formalism for the specification of trust or any method to address the dependence of trust on time. There are no methods for measuring trust, comparing trust values, and composing trust values.

The idea of a new formal model to assess multiple levels of trust, which is more in keeping with the social models of trust used by policy makers, has been at the heart of our on-going research during the past three years. In the first year of the project the core idea of a new “Vector” model of trust was formulated and the key model elements were identified. In the second year the theoretical aspects of the model were developed. The “Vector” model of trust provides an objective decision support system, determines trust values, facilitates comparing trust values of two systems, negotiates and manages trust values, and computes the trust value of composed systems (given the trust values of the component subsystems). The model elements are as follows:

1. A trust (distrust) relationship between a truster and a trustee is a three-element vector of numeric values:

   - Experience - e.g. history
   - Knowledge - e.g. specifications and properties
   - Recommendation - e.g. personal reports

2. Trust comparison and composition are accomplished using the three-element vector of numeric values in 1 above.

3. Trust depends on previously established trust values and can change with time if not updated.

4. A trust management system formulates, stores and manages trust vectors.

In the third year of the project, we subjected the model to peer-review. The process has resulted in several papers in reputed conferences. In the third year of the project, we subjected the model to peer-review. The process has resulted in several papers in reputed conferences.

Based partially on the feedback from the peer-review process, our own perceptions and comments from potential users of the model, we have identified three core areas in which the model needs to be refined and extended. These extensions will enable the model to be interoperable across different domains.

A semantically richer representation of trust contexts The current model can reason about trust relationships only within the confines of a single context. It allows two or more trust values to be compared or combined only when there is an exact match on the context. Even if semantically equivalent contexts are expressed differently for two trust relationships, the model is unable to compare or compose them. For example, let a user *A* trust user *C* to a degree *T* to keep a piece of information confidential (the context). Let another user *B* trust *C* to degree *S* to keep the same piece of information secret (context for this case). Although semantically the two contexts are equivalent the current model fails to compare the trust degrees *T* and *S* because it cannot interpret the strings "confidential" and "secret" or identify the relationship between the same. This constraint is too restrictive and may not be realistic in most situations. We need to refine the current model to handle this scenario.

An ability to extrapolate trust relationships The model fails to evaluate a trust relationship between a truster and a trustee if the former does not have any experience, knowledge or recommendation (the model parameters) about the latter within a given context. This is still the

case even if the truster has useful information about the model parameters in a related context. For example, let an organization *A* (the truster) trust a software developer *B* (the trustee) to a degree *T* to produce excellent quality anti-virus software (the context). It appears natural for *A* to try to determine how much to trust *B* to produce application level firewall software (a related context) based solely on the available information. However, the current model returns an undefined value in such a case. The model, thus, needs an ability to capture the semantic relationship between the two contexts and then extrapolate one trust relationship based on the semantic relationship between the two contexts.

An ability to reason about trust chains The current model cannot successfully evaluate trust chains. This is because trust relationships under the current model are not considered transitive in nature. However, certain types of trust relationships in real world are indeed transitive. The best example of this is found in the delegation process. Suppose a user *C* trusts another user *D* to determine who has access to *C*Õs resources. Thus if *D* trusts a third user *F* and allows *F* to access one of *C*Õs resources, *C* transitively trusts *F* to some degree not to corrupt that resource in a malicious manner. Trust chains also arise in dynamic ad-hoc coalitions. During the formation of such a coalition it is necessary to trust (to some extent) a new entrant who is being introduced to the group by an existing member. The current model needs to be extended to handle trust chains.

With these extensions in place, the refined model will be significantly more expressive and usable than the current model. A major task remains after that, namely, a proper validation of the model on a real word application. We hope to get future support from the AFRL in our effort to refine and validate the model in this manner.

Publications Resulting from Project

The project resulted in the following papers.

1. Indrajit Ray and Sudip Chakraborty, “A Vector Model of Trust for Developing Trustworthy Systems,” In Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS ’04), Sophia Antipolis, France, September 2004.

2. Indrajit Ray, Sudip Chakraborty and Indrakshi Ray, “VTrust: A Trust Management System Based on a Vector Model of Trust,” In Proceedings of the 1st International Conference on Information Systems Security (ICISS ’05), Kolkata, India, December 2005.

3. Anna C. Squicciarini, Elisa Bertino, Elena Ferrari, and Indrakshi Ray, " Achieving Privacy with an Ontology-Based Approach in Trust Negotiations," IEEE Transactions on Dependable and Secure Computing, 3(1), January-March, 2006.

4. Siv Hilde Houmb, Indrakshi Ray, and Indrajit Ray," Estimating the Relative Trustworthiness of Information Sources in Security Solution Evaluation," In Proceedings of the 4th International Conference on Trust Management, Pisa, Italy, May 2006.

5. Sudip Chakraborty and Indrajit Ray, "TrustBAC - Integrating Trust Relationships into the RBAC Model for Access Control in Open Systems," In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT'06), Lake Tahoe, CA, USA, June 7-9, 2006.

6. Sudip Chakraborty, and Indrajit Ray, " Allowing Finer Control Over Privacy Using Trust as a Benchmark," In Proceedings of the 7th Annual IEEE Information Assurance Workshop (IAW'06), United States Military Academy, West Point, NY, June 21-23, 2006.

7. Indrajit Ray and Sudip Chakraborty, "A Framework for Flexible Access Control in Digital Library Systems," In Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'06), SAP Labs, Sophia Antipolis, France, July 31-August 2, 2006.

Bibliography

[1] M. Blaze, J. Feigenbaum, and J. Ioannidis. The KeyNote Trust Management System Version 2. Internet Society, Network Working Group. RFC 2704, 1999.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1996.

[3] D.W. Chadwick, A. Otenko, and E. Ball. Role-Based Access Control with X.509 Attribute Certificates. *IEEE Internet Computing*, 7(2):62–69, March/April 2003.

[4] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, August 2001.

[5] T. Grandison and M. Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys and Tutorials*, 3(4), Fourth Quarter 2000.

[6] A.O. Hirschman. Three Ways of Compilcating Some Categories of Economic Discourse. *American Economic Review*, 74(2), 1984.

[7] N. Li and J.C. Mitchell. Datalog with Constraints: A Foundation for Trust-management Languages. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, New Orleans, Louisiana, January 2003.

[8] N. Li and J.C. Mitchell. RT: A Role-based Trust Management Framework. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, Washington D.C., April 2003.

[9] N. Li, J.C. Mitchell, and W.H. Winsborough. Design of a Role-Based Trust-Management Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Oakland, California, May 2002.

[10] N. Li, W.H. Winsborough, and J.C. Mitchell. Beyond Proof-of-Compliance: Safety and Availability Analysis in Trust Management. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, Oakland, California, May 2003.